autogluon_each_location

October 28, 2023

1 Config

[95]:
```
# config

label = 'y'
metric = 'mean_absolute_error'
time_limit = None
presets = "best_quality"#'best_quality'

do_drop_ds = True
# hour, dayofweek, dayofmonth, month, year
use_dt_attrs = []#["hour", "year"]
use_estimated_diff_attr = False
use_is_estimated_attr = True

drop_night_outliers = True
drop_null_outliers = False

# to_drop = ["snow_drift:idx", "snow_density:kgm3", "wind_speed_w_1000hPa:ms",␣
↪"dew_or_rime:idx", "prob_rime:p", "fresh_snow_12h:cm", "fresh_snow_24h:cm",␣
↪"wind_speed_u_10m:ms", "wind_speed_v_10m:ms", "snow_melt_10min:mm",␣
↪"rain_water:kgm2", "dew_point_2m:K", "precip_5min:mm", "absolute_humidity_2m:
↪gm3", "air_density_2m:kgm3"]#, "msl_pressure:hPa", "pressure_50m:hPa", ␣
↪"pressure_100m:hPa"]
to_drop = ["wind_speed_w_1000hPa:ms", "wind_speed_u_10m:ms", "wind_speed_v_10m:
↪ms"]


excluded_model_types = ['CAT', 'XGB', 'RF']


use_groups = False
n_groups = 8

# auto_stack = True
num_stack_levels = 1
num_bag_folds = None# 8
```

```
num_bag_sets = None#20

use_tune_data = True
use_test_data = True
#tune_and_test_length = 0.5 # 3 months from end
# holdout_frac = None
use_bag_holdout = True # Enable this if there is a large gap between score_val␣
↪and score_test in stack models.

sample_weight = None#'sample_weight' #None
weight_evaluation = False#
sample_weight_estimated = 1
sample_weight_may_july = 1

run_analysis = False

shift_predictions_by_average_of_negatives_then_clip = False
clip_predictions = True
shift_predictions = False
```

2 Loading and preprocessing

[96]:
```
import pandas as pd
import numpy as np



import warnings
warnings.filterwarnings("ignore")


def feature_engineering(X):
    # shift all columns with "1h" in them by 1 hour, so that for index 16:00,␣
↪we have the values from 17:00
    # but only for the columns with "1h" in the name
    #X_shifted = X.filter(regex="\dh").shift(-1, axis=1)
    #print(f"Number of columns with 1h in name: {X_shifted.columns}")


    columns = ['clear_sky_energy_1h:J', 'diffuse_rad_1h:J', 'direct_rad_1h:J',
               'fresh_snow_12h:cm', 'fresh_snow_1h:cm', 'fresh_snow_24h:cm',
               'fresh_snow_3h:cm', 'fresh_snow_6h:cm']

    # Filter rows where index.minute == 0
    X_shifted = X[X.index.minute == 0][columns].copy()
```

```
    # Create a set for constant-time lookup
    index_set = set(X.index)

    # Vectorized time shifting
    one_hour = pd.Timedelta('1 hour')
    shifted_indices = X_shifted.index + one_hour
    X_shifted.loc[shifted_indices.isin(index_set)] = X.
↪loc[shifted_indices[shifted_indices.isin(index_set)]][columns]

    # set last row to same as second last row
    X_shifted.iloc[-1] = X_shifted.iloc[-2]

    # Count
    count1 = len(shifted_indices[shifted_indices.isin(index_set)])
    count2 = len(X_shifted) - count1

    print("COUNT1", count1)
    print("COUNT2", count2)

    # Rename columns
    X_old_unshifted = X_shifted.copy()
    X_old_unshifted.columns = [f"{col}_not_shifted" for col in X_old_unshifted.
↪columns]

    date_calc = None
    # If 'date_calc' is present, handle it
    if 'date_calc' in X.columns:
        date_calc = X[X.index.minute == 0]['date_calc']




    # resample to hourly
    print("index: ", X.index[0])
    X = X.resample('H').mean()
    print("index AFTER: ", X.index[0])

    X[columns] = X_shifted[columns]
    #X[X_old_unshifted.columns] = X_old_unshifted

    if date_calc is not None:
        X['date_calc'] = date_calc

    return X
```

```
def fix_X(X, name):
    # Convert 'date_forecast' to datetime format and replace original column␣
↪with 'ds'
    X['ds'] = pd.to_datetime(X['date_forecast'])
    X.drop(columns=['date_forecast'], inplace=True, errors='ignore')
    X.sort_values(by='ds', inplace=True)
    X.set_index('ds', inplace=True)


    X = feature_engineering(X)

    return X




def handle_features(X_train_observed, X_train_estimated, X_test, y_train):
    X_train_observed = fix_X(X_train_observed, "X_train_observed")
    X_train_estimated = fix_X(X_train_estimated, "X_train_estimated")
    X_test = fix_X(X_test, "X_test")


    if weight_evaluation:
        # add sample weights, which are 1 for observed and 3 for estimated
        X_train_observed["sample_weight"] = 1
        X_train_estimated["sample_weight"] = sample_weight_estimated
        X_test["sample_weight"] = sample_weight_estimated


    y_train['ds'] = pd.to_datetime(y_train['time'])
    y_train.drop(columns=['time'], inplace=True)
    y_train.sort_values(by='ds', inplace=True)
    y_train.set_index('ds', inplace=True)

    return X_train_observed, X_train_estimated, X_test, y_train





def preprocess_data(X_train_observed, X_train_estimated, X_test, y_train,␣
↪location):
    # convert to datetime
    X_train_observed, X_train_estimated, X_test, y_train =␣
↪handle_features(X_train_observed, X_train_estimated, X_test, y_train)

    if use_estimated_diff_attr:
        X_train_observed["estimated_diff_hours"] = 0
```

```
        X_train_estimated["estimated_diff_hours"] = (X_train_estimated.index - pd.to_datetime(X_train_estimated["date_calc"])).dt.total_seconds() / 3600
        X_test["estimated_diff_hours"] = (X_test.index - pd.to_datetime(X_test["date_calc"])).dt.total_seconds() / 3600

        X_train_estimated["estimated_diff_hours"] = X_train_estimated["estimated_diff_hours"].astype('int64')
        # the filled once will get dropped later anyways, when we drop y nans
        X_test["estimated_diff_hours"] = X_test["estimated_diff_hours"].fillna(-50).astype('int64')

    if use_is_estimated_attr:
        X_train_observed["is_estimated"] = 0
        X_train_estimated["is_estimated"] = 1
        X_test["is_estimated"] = 1


    # drop date_calc
    X_train_estimated.drop(columns=['date_calc'], inplace=True)
    X_test.drop(columns=['date_calc'], inplace=True)


    y_train["y"] = y_train["pv_measurement"].astype('float64')
    y_train.drop(columns=['pv_measurement'], inplace=True)
    X_train = pd.concat([X_train_observed, X_train_estimated])


    # clip all y values to 0 if negative
    y_train["y"] = y_train["y"].clip(lower=0)

    X_train = pd.merge(X_train, y_train, how="inner", left_index=True, right_index=True)

    # print number of nans in y
    print(f"Number of nans in y: {X_train['y'].isna().sum()}")

    print(f"Size of estimated after dropping nans: {len(X_train[X_train['is_estimated']==1].dropna(subset=['y']))}")


    X_train["location"] = location
    X_test["location"] = location

    return X_train, X_test
# Define locations
locations = ['A', 'B', 'C']
```

```python
X_trains = []
X_tests = []
# Loop through locations
for loc in locations:
    print(f"Processing location {loc}...")
    # Read target training data
    y_train = pd.read_parquet(f'{loc}/train_targets.parquet')

    # Read estimated training data and add location feature
    X_train_estimated = pd.read_parquet(f'{loc}/X_train_estimated.parquet')

    # Read observed training data and add location feature
    X_train_observed= pd.read_parquet(f'{loc}/X_train_observed.parquet')

    # Read estimated test data and add location feature
    X_test_estimated = pd.read_parquet(f'{loc}/X_test_estimated.parquet')

    # Preprocess data
    X_train, X_test = preprocess_data(X_train_observed, X_train_estimated,␣
 ↪X_test_estimated, y_train, loc)

    X_trains.append(X_train)
    X_tests.append(X_test)

# Concatenate all data and save to csv
X_train = pd.concat(X_trains)
X_test = pd.concat(X_tests)
```

```
Processing location A…
COUNT1 29667
COUNT2 1
index:  2019-06-02 22:00:00
index AFTER:  2019-06-02 22:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 0
Size of estimated after dropping nans: 4418
Processing location B…
COUNT1 29232
COUNT2 1
index:  2019-01-01 00:00:00
```

```
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 4
Size of estimated after dropping nans: 3625
Processing location C…
COUNT1 29206
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 6059
Size of estimated after dropping nans: 2954
```

2.1 Feature enginering

2.1.1 Remove anomalies

[97]:
```python
import numpy as np
import pandas as pd


# loop thorugh x train[y], keep track of streaks of same values and replace␣
↪them with nan if they are too long
# also replace nan with 0

import numpy as np

def replace_streaks_with_nan(df, max_streak_length, column="y"):
    for location in df["location"].unique():
        x = df[df["location"] == location][column].copy()

        last_val = None
        streak_length = 1
        streak_indices = []
```

```
        allowed = [0]
        found_streaks = {}

        for idx in x.index:
            value = x[idx]
            # if location == "B":
            #     continue

            if value == last_val and value not in allowed:
                streak_length += 1
                streak_indices.append(idx)
            else:
                streak_length = 1
                last_val = value
                streak_indices.clear()

            if streak_length > max_streak_length:
                found_streaks[value] = streak_length

                for streak_idx in streak_indices:
                    x[idx] = np.nan
                streak_indices.clear()  # clear after setting to NaN to avoid␣
↪setting multiple times
        df.loc[df["location"] == location, column] = x

        print(f"Found streaks for location {location}: {found_streaks}")

    return df


# deep copy of X_train into x_copy
X_train = replace_streaks_with_nan(X_train.copy(), 3, "y")
```

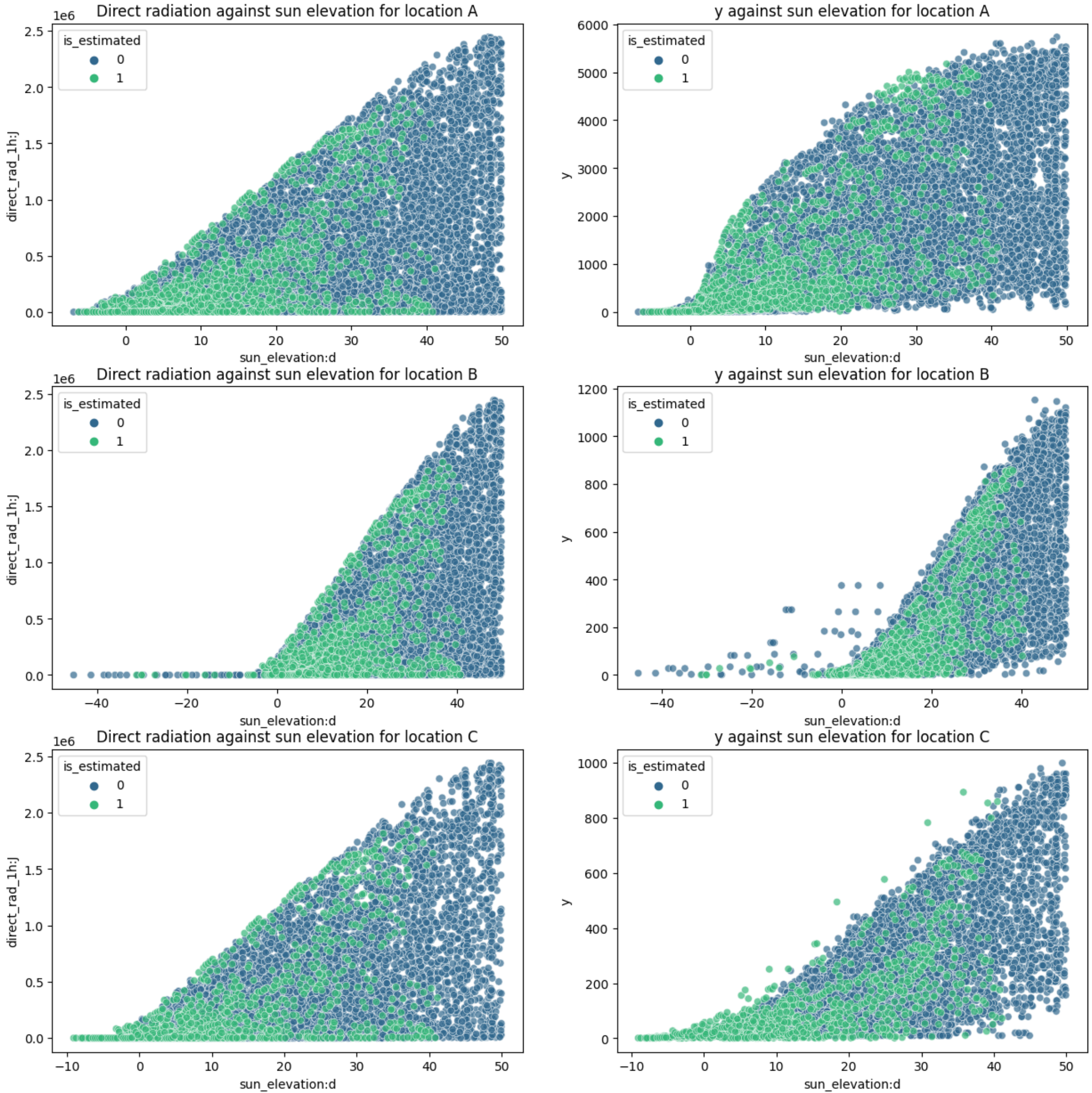
```
Found streaks for location A: {}
Found streaks for location B: {3.45: 28, 6.9: 7, 12.9375: 5, 13.8: 8, 276.0: 78,
18.975: 58, 0.8625: 4, 118.1625: 33, 34.5: 11, 183.7125: 1058, 87.1125: 7,
79.35: 34, 7.7625: 12, 27.6: 448, 273.41249999999997: 72, 264.78749999999997:
55, 169.05: 33, 375.1875: 56, 314.8125: 66, 76.7625: 10, 135.4125: 216, 81.9375:
202, 2.5875: 12, 81.075: 210}
Found streaks for location C: {9.8: 4, 29.400000000000002: 4, 19.6: 4}
```

[98]:
```
# print num rows
temprows = len(X_train)
X_train.dropna(subset=['y', 'direct_rad_1h:J', 'diffuse_rad_1h:J'],␣
↪inplace=True)
print("Dropped rows: ", temprows - len(X_train))
```

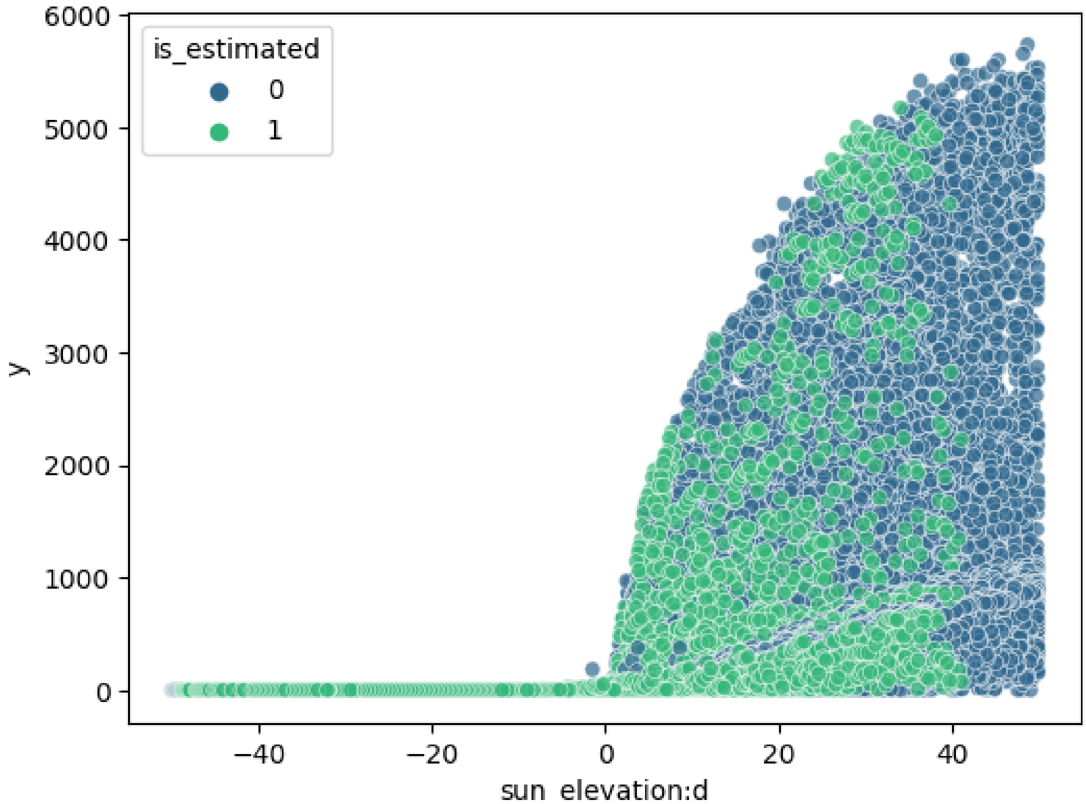
```
Dropped rows:  9293
```

[99]:

```
import matplotlib.pyplot as plt
import seaborn as sns
# Filter out rows where y == 0
temp = X_train[X_train["y"] != 0]

# Plotting
fig, axes = plt.subplots(len(locations), 2, figsize=(15, 5 * len(locations)))

for idx, location in enumerate(locations):
    sns.scatterplot(ax=axes[idx][0], data=temp[temp["location"] == location],␣
 ↪x="sun_elevation:d", y="direct_rad_1h:J", hue="is_estimated",␣
 ↪palette="viridis", alpha=0.7)
    axes[idx][0].set_title(f"Direct radiation against sun elevation for␣
 ↪location {location}")

    sns.scatterplot(ax=axes[idx][1], data=temp[temp["location"] == location],␣
 ↪x="sun_elevation:d", y="y", hue="is_estimated", palette="viridis", alpha=0.7)
    axes[idx][1].set_title(f"y against sun elevation for location {location}")

# plt.tight_layout()
# plt.show()
```

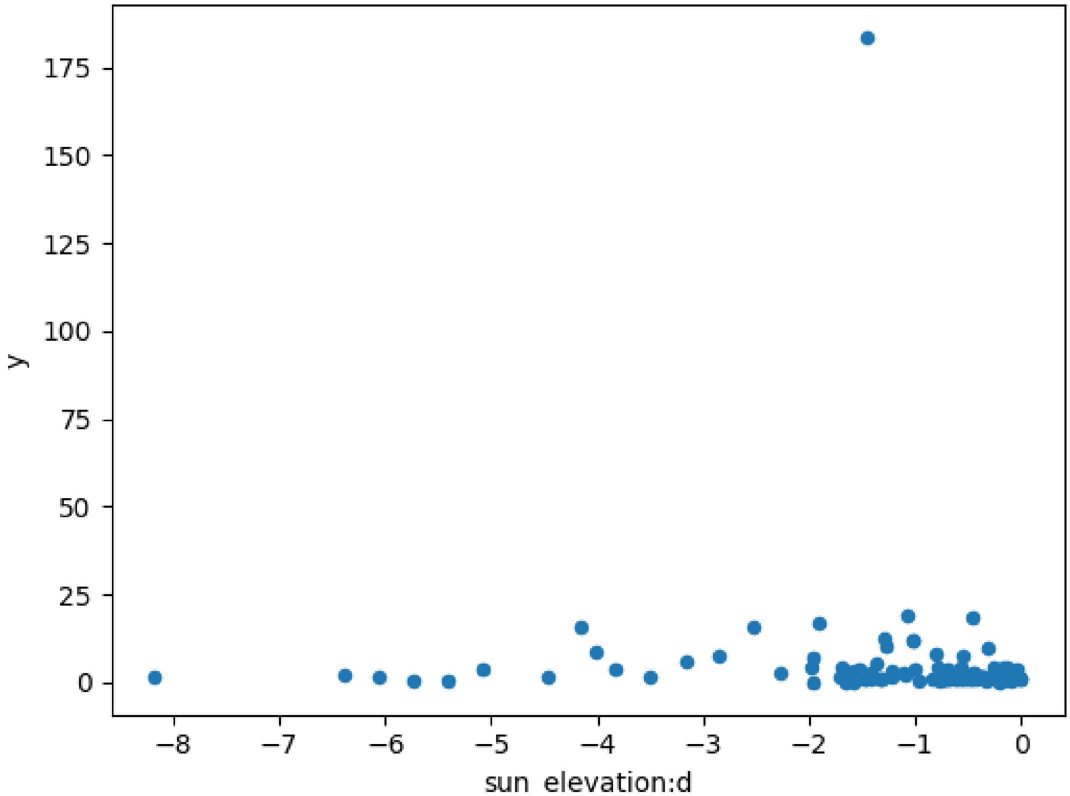


[100]:
```python
thresh = 0.1

# Update "y" values to NaN if they don't meet the criteria
mask = (X_train["direct_rad_1h:J"] <= thresh) & (X_train["diffuse_rad_1h:J"] <=␣
 ↪thresh) & (X_train["y"] >= 0.1)
if drop_night_outliers:
    X_train.loc[mask, "y"] = np.nan

# Plot using sns scatterplot
sns.scatterplot(data=X_train, x="sun_elevation:d", y="y", hue="is_estimated",␣
 ↪palette="viridis", alpha=0.7)
plt.show()
```



[101]:
```
# location B count number of rows with y > 0 and sun_elevation:d < 0

condition = (X_train["location"] == "B") & (X_train["y"] > 0) &␣
 ↪(X_train["sun_elevation:d"] < 0)
bad = X_train[condition]

bad.plot.scatter(x="sun_elevation:d", y="y")
```

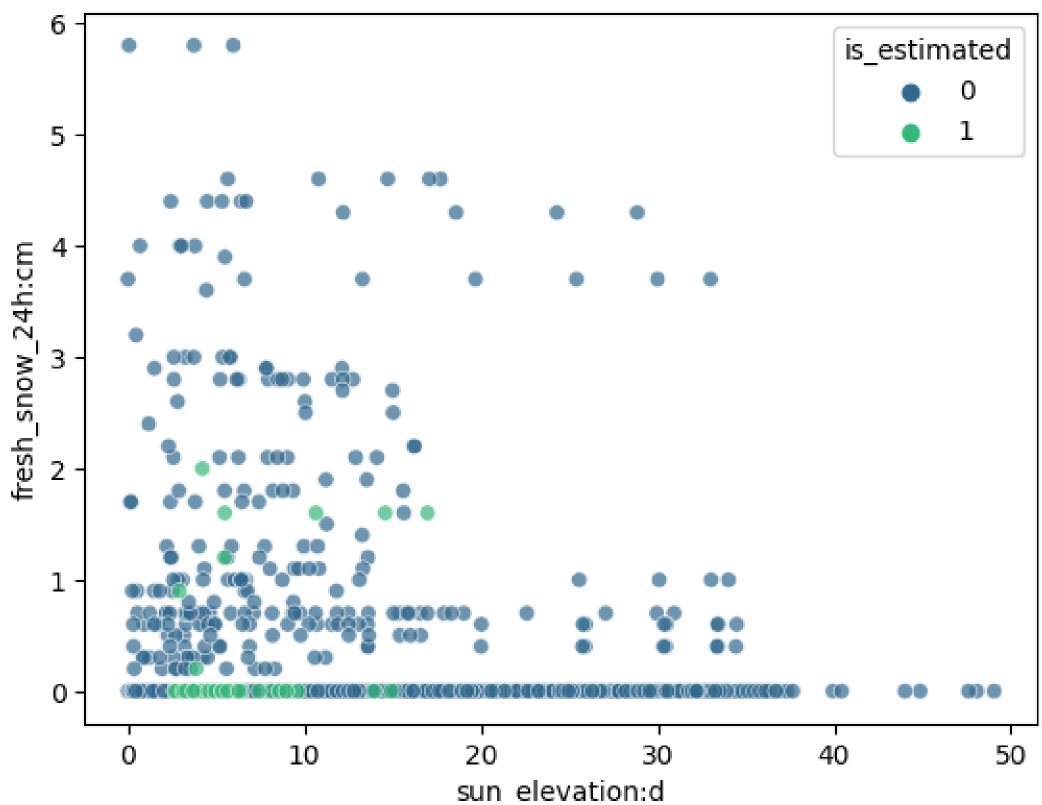
[101]: <AxesSubplot: xlabel='sun_elevation:d', ylabel='y'>



[102]:
```
# set y to nan where y is 0, but direct_rad_1h:J or diffuse_rad_1h:J are > 0␣
↪(or some threshold)
threshold_direct = X_train["direct_rad_1h:J"].max() * 0.01
threshold_diffuse = X_train["diffuse_rad_1h:J"].max() * 0.01
print(f"Threshold direct: {threshold_direct}")
print(f"Threshold diffuse: {threshold_diffuse}")

mask = (X_train["y"] == 0) & ((X_train["direct_rad_1h:J"] > threshold_direct) |␣
↪(X_train["diffuse_rad_1h:J"] > threshold_diffuse)) & (X_train["sun_elevation:
↪d"] > 0) & (X_train["fresh_snow_24h:cm"] < 6) & (X_train[['fresh_snow_12h:
↪cm', 'fresh_snow_1h:cm',  'fresh_snow_3h:cm', 'fresh_snow_6h:cm']].
↪sum(axis=1) == 0)
print(len(X_train[mask]))

#print(X_train[mask][[x for x in X_train.columns if "snow" in x]])

# show plot where mask is true
#sns.scatterplot(data=X_train[mask], x="sun_elevation:d", y="y",␣
↪hue="is_estimated", palette="viridis", alpha=0.7)
```

```
sns.scatterplot(data=X_train[mask], x="sun_elevation:d", y="fresh_snow_24h:cm",␣
↪hue="is_estimated", palette="viridis", alpha=0.7)
plt.show()

#sns.scatterplot(data=X_train[mask], x="fresh_snow_24h:cm",␣
↪y="total_cloud_cover:p", hue="is_estimated", palette="viridis", alpha=0.7)



# set y to nan where mask
if drop_null_outliers:
    X_train.loc[mask, "y"] = np.nan

# show how many rows for each location, and for estimated and not estimated
X_train[mask].groupby(["location", "is_estimated"]).count()["direct_rad_1h:J"]
```

```
Threshold direct: 24458.97
Threshold diffuse: 11822.505000000001
2599
```



[102]:
```
location  is_estimated
A         0                 87
          1                 10
B         0               1250
          1                 32
C         0               1174
          1                 46
Name: direct_rad_1h:J, dtype: int64
```

[103]:
```python
# print num rows
temprows = len(X_train)
X_train.dropna(subset=['y', 'direct_rad_1h:J', 'diffuse_rad_1h:J'],␣
 ↪inplace=True)
print("Dropped rows: ", temprows - len(X_train))
```

```
Dropped rows:  1876
```

2.1.2 Other stuff

[104]:
```python
import numpy as np
import pandas as pd

for attr in use_dt_attrs:
    X_train[attr] = getattr(X_train.index, attr)
    X_test[attr] = getattr(X_test.index, attr)

#print(X_train.head())


# If the "sample_weight" column is present and weight_evaluation is True,␣
 ↪multiply sample_weight with sample_weight_may_july if the ds is between␣
 ↪05-01 00:00:00 and 07-03 23:00:00, else add sample_weight as a column to␣
 ↪X_train
if weight_evaluation:
    if "sample_weight" not in X_train.columns:
        X_train["sample_weight"] = 1

    X_train.loc[((X_train.index.month >= 5) & (X_train.index.month <= 6)) |␣
 ↪((X_train.index.month == 7) & (X_train.index.day <= 3)), "sample_weight"] *=␣
 ↪sample_weight_may_july


print(X_train.iloc[200])
print(X_train[((X_train.index.month >= 5) & (X_train.index.month <= 6)) |␣
 ↪((X_train.index.month == 7) & (X_train.index.day <= 3))].head(1))
```

```
if use_groups:
    # fix groups for cross validation
    locations = X_train['location'].unique()  # Assuming 'location' is the name␣
 ↪of the column representing locations

    grouped_dfs = []  # To store data frames split by location

    # Loop through each unique location
    for loc in locations:
        loc_df = X_train[X_train['location'] == loc]

        # Sort the DataFrame for this location by the time column
        loc_df = loc_df.sort_index()

        # Calculate the size of each group for this location
        group_size = len(loc_df) // n_groups

        # Create a new 'group' column for this location
        loc_df['group'] = np.repeat(range(n_groups),␣
 ↪repeats=[group_size]*(n_groups-1) + [len(loc_df) - group_size*(n_groups-1)])

        # Append to list of grouped DataFrames
        grouped_dfs.append(loc_df)

    # Concatenate all the grouped DataFrames back together
    X_train = pd.concat(grouped_dfs)
    X_train.sort_index(inplace=True)
    print(X_train["group"].head())





X_train.drop(columns=to_drop, inplace=True)
X_test.drop(columns=to_drop, inplace=True)

X_train.to_csv('X_train_raw.csv', index=True)
X_test.to_csv('X_test_raw.csv', index=True)
```

```
absolute_humidity_2m:gm3              7.625
air_density_2m:kgm3                  1.2215
ceiling_height_agl:m            3644.050049
clear_sky_energy_1h:J            2896336.75
clear_sky_rad:W                  753.849976
cloud_base_agl:m                3644.050049
dew_or_rime:idx                         0.0
```

```
dew_point_2m:K                     280.475006
diffuse_rad:W                      127.475006
diffuse_rad_1h:J                   526032.625
direct_rad:W                            488.0
direct_rad_1h:J                   1718048.625
effective_cloud_cover:p             18.200001
elevation:m                               6.0
fresh_snow_12h:cm                         0.0
fresh_snow_1h:cm                          0.0
fresh_snow_24h:cm                         0.0
fresh_snow_3h:cm                          0.0
fresh_snow_6h:cm                          0.0
is_day:idx                                1.0
is_in_shadow:idx                          0.0
msl_pressure:hPa                  1026.775024
precip_5min:mm                            0.0
precip_type_5min:idx                      0.0
pressure_100m:hPa                 1013.599976
pressure_50m:hPa                  1019.599976
prob_rime:p                               0.0
rain_water:kgm2                           0.0
relative_humidity_1000hPa:p         53.825001
sfc_pressure:hPa                  1025.699951
snow_density:kgm3                         NaN
snow_depth:cm                             0.0
snow_drift:idx                            0.0
snow_melt_10min:mm                        0.0
snow_water:kgm2                           0.0
sun_azimuth:d                      222.089005
sun_elevation:d                     44.503498
super_cooled_liquid_water:kgm2            0.0
t_1000hPa:K                        286.700012
total_cloud_cover:p                 18.200001
visibility:m                         52329.25
wind_speed_10m:ms                         2.6
wind_speed_u_10m:ms                      -1.9
wind_speed_v_10m:ms                     -1.75
wind_speed_w_1000hPa:ms                   0.0
is_estimated                                0
y                                     4367.44
location                                    A
Name: 2019-06-11 13:00:00, dtype: object
                     absolute_humidity_2m:gm3  air_density_2m:kgm3  \
ds
2019-06-02 23:00:00                       7.7               1.2235

                     ceiling_height_agl:m  clear_sky_energy_1h:J  \
ds
```

```
2019-06-02 23:00:00           1689.824951                   0.0

                     clear_sky_rad:W  cloud_base_agl:m  dew_or_rime:idx  \
ds
2019-06-02 23:00:00              0.0       1689.824951              0.0

                     dew_point_2m:K  diffuse_rad:W  diffuse_rad_1h:J  …  \
ds                                                                     …
2019-06-02 23:00:00      280.299988            0.0               0.0  …

                     t_1000hPa:K  total_cloud_cover:p  visibility:m  \
ds
2019-06-02 23:00:00   286.899994                100.0  33770.648438

                     wind_speed_10m:ms  wind_speed_u_10m:ms  \
ds
2019-06-02 23:00:00               3.35                -3.35

                     wind_speed_v_10m:ms  wind_speed_w_1000hPa:ms  \
ds
2019-06-02 23:00:00                0.275                      0.0

                     is_estimated    y  location
ds
2019-06-02 23:00:00             0  0.0         A

[1 rows x 48 columns]
```

[105]:
```
# Create a plot of X_train showing its "y" and color it based on the value of the sample_weight column.
if "sample_weight" in X_train.columns:
    import matplotlib.pyplot as plt
    import seaborn as sns
    sns.scatterplot(data=X_train, x=X_train.index, y="y", hue="sample_weight", palette="deep", size=3)
    plt.show()
```

[106]:
```
def normalize_sample_weights_per_location(df):
    for loc in locations:
        loc_df = df[df["location"] == loc]
        loc_df["sample_weight"] = loc_df["sample_weight"] / loc_df["sample_weight"].sum() * loc_df.shape[0]
        df[df["location"] == loc] = loc_df
    return df

import pandas as pd
```

```python
def split_and_shuffle_data(input_data, num_bins, frac1):
    """
    Splits the input_data into num_bins and shuffles them, then divides the␣
↪bins into two datasets based on the given fraction for the first set.

    Args:
        input_data (pd.DataFrame): The data to be split and shuffled.
        num_bins (int): The number of bins to split the data into.
        frac1 (float): The fraction of each bin to go into the first output␣
↪dataset.

    Returns:
        pd.DataFrame, pd.DataFrame: The two output datasets.
    """
    # Validate the input fraction
    if frac1 < 0 or frac1 > 1:
        raise ValueError("frac1 must be between 0 and 1.")

    if frac1==1:
        return input_data, pd.DataFrame()

    # Calculate the fraction for the second output set
    frac2 = 1 - frac1

    # Calculate bin size
    bin_size = len(input_data) // num_bins

    # Initialize empty DataFrames for output
    output_data1 = pd.DataFrame()
    output_data2 = pd.DataFrame()

    for i in range(num_bins):
        # Shuffle the data in the current bin
        np.random.seed(i)
        current_bin = input_data.iloc[i * bin_size: (i + 1) * bin_size].
↪sample(frac=1)

        # Calculate the sizes for each output set
        size1 = int(len(current_bin) * frac1)

        # Split and append to output DataFrames
        output_data1 = pd.concat([output_data1, current_bin.iloc[:size1]])
        output_data2 = pd.concat([output_data2, current_bin.iloc[size1:]])

    # Shuffle and split the remaining data
    remaining_data = input_data.iloc[num_bins * bin_size:].sample(frac=1)
```

```
    remaining_size1 = int(len(remaining_data) * frac1)

    output_data1 = pd.concat([output_data1, remaining_data.iloc[:
↪remaining_size1]])
    output_data2 = pd.concat([output_data2, remaining_data.iloc[remaining_size1:
↪]])

    return output_data1, output_data2
```

[107]:
```
from autogluon.tabular import TabularDataset, TabularPredictor
data = TabularDataset('X_train_raw.csv')
# set group column of train_data be increasing from 0 to 7 based on time, the␣
↪first 1/8 of the data is group 0, the second 1/8 of the data is group 1, etc.
data['ds'] = pd.to_datetime(data['ds'])
data = data.sort_values(by='ds')

# # print size of the group for each location
# for loc in locations:
#     print(f"Location {loc}:")
#     print(train_data[train_data["location"] == loc].groupby('group').size())


# get end date of train data and subtract 3 months
#split_time = pd.to_datetime(train_data["ds"]).max() - pd.
↪Timedelta(hours=tune_and_test_length)
# 2022-10-28 22:00:00
split_time = pd.to_datetime("2022-10-28 22:00:00")
train_set = TabularDataset(data[data["ds"] < split_time])
estimated_set = TabularDataset(data[data["ds"] >= split_time]) # only estimated

test_set = pd.DataFrame()
tune_set = pd.DataFrame()
new_train_set = pd.DataFrame()

if not use_tune_data:
    raise Exception("Not implemented")

for location in locations:
    loc_data = data[data["location"] == location]
    num_train_rows = len(loc_data)

    tune_rows = 1500.0 # 2500.0
    if use_test_data:
        tune_rows = 1880.0#max(3000.0,␣
↪len(estimated_set[estimated_set["location"] == location]))
```

```
    holdout_frac = max(0.01, min(0.1, tune_rows / num_train_rows)) * num_train_rows / len(estimated_set[estimated_set["location"] == location])

    print(f"Size of estimated for location {location}: {len(estimated_set[estimated_set['location'] == location])}. Holdout frac should be % of estimated: {holdout_frac}")

    # shuffle and split data
    loc_tune_set, loc_new_train_set = split_and_shuffle_data(estimated_set[estimated_set['location'] == location], 40, holdout_frac)
    print(f"Length of location tune set : {len(loc_tune_set)}")
    new_train_set = pd.concat([new_train_set, loc_new_train_set])

    if use_test_data:
        loc_test_set, loc_tune_set = split_and_shuffle_data(loc_tune_set, 40, 0.2)
        test_set = pd.concat([test_set, loc_test_set])

    tune_set = pd.concat([tune_set, loc_tune_set])




print("Length of train set before adding test set", len(train_set))
# add rest to train_set
train_set = pd.concat([train_set, new_train_set])
print("Length of train set after adding test set", len(train_set))




if use_groups:
    test_set = test_set.drop(columns=['group'])


tuning_data = tune_set

# number of rows in tuning data for each location
print("Shapes of tuning data", tuning_data.groupby('location').size())



if use_test_data:
    test_data = test_set
    print("Shape of test", test_data.shape[0])
```

```
train_data = train_set

# ensure sample weights for your training (or tuning) data sum to the number of␣
↪rows in the training (or tuning) data.
if weight_evaluation:
    # ensure sample weights for data sum to the number of rows in the tuning /
↪train data.
    tuning_data = normalize_sample_weights_per_location(tuning_data)
    train_data = normalize_sample_weights_per_location(train_data)
    if use_test_data:
        test_data = normalize_sample_weights_per_location(test_data)


train_data = TabularDataset(train_data)
tuning_data = TabularDataset(tuning_data)

if use_test_data:
    test_data = TabularDataset(test_data)
```

```
Loaded data from: X_train_raw.csv | Columns = 46 / 46 | Rows = 87917 -> 87917
```

```
Size of estimated for location A: 4214. Holdout frac should be % of estimated:
0.4461319411485524
Length of location tune set : 1846
Size of estimated for location B: 3533. Holdout frac should be % of estimated:
0.5321256722332296
Length of location tune set : 1846
Size of estimated for location C: 2923. Holdout frac should be % of estimated:
0.6431748203900103
Length of location tune set : 1841
Length of train set before adding test set 77247
Length of train set after adding test set 82384
Shapes of tuning data location
A    1485
B    1485
C    1481
dtype: int64
Shape of test 1082
```

3 Quick EDA

[108]:
```
if run_analysis:
    import autogluon.eda.auto as auto
    auto.dataset_overview(train_data=train_data, test_data=test_data,␣
↪label="y", sample=None)
```

[109]:
```python
if run_analysis:
    auto.target_analysis(train_data=train_data, label="y", sample=None)
```

4 Modeling

[110]:
```python
import os


# Get the last submission number
last_submission_number = int(max([int(filename.split('_')[1].split('.')[0]) for␣
 ↪filename in os.listdir('submissions') if "submission" in filename]))
print("Last submission number:", last_submission_number)
print("Now creating submission number:", last_submission_number + 1)

# Create the new filename
new_filename = f'submission_{last_submission_number + 1}'

hello = os.environ.get('HELLO')
if hello is not None:
    new_filename += f'_{hello}'

print("New filename:", new_filename)
```

```
Last submission number: 109
Now creating submission number: 110
New filename: submission_110_jorge
```

[111]:
```python
predictors = [None, None, None]
```

[112]:
```python
def fit_predictor_for_location(loc):
    print(f"Training model for location {loc}...")
    # sum of sample weights for this location, and number of rows, for both␣
 ↪train and tune data and test data
    if weight_evaluation:
        print("Train data sample weight sum:",␣
 ↪train_data[train_data["location"] == loc]["sample_weight"].sum())
        print("Train data number of rows:", train_data[train_data["location"]␣
 ↪== loc].shape[0])
        if use_tune_data:
            print("Tune data sample weight sum:",␣
 ↪tuning_data[tuning_data["location"] == loc]["sample_weight"].sum())
            print("Tune data number of rows:",␣
 ↪tuning_data[tuning_data["location"] == loc].shape[0])
        if use_test_data:
            print("Test data sample weight sum:",␣
 ↪test_data[test_data["location"] == loc]["sample_weight"].sum())
```

```
            print("Test data number of rows:", test_data[test_data["location"]␣
 ↪== loc].shape[0])
    predictor = TabularPredictor(
        label=label,
        eval_metric=metric,
        path=f"AutogluonModels/{new_filename}_{loc}",
        # sample_weight=sample_weight,
        # weight_evaluation=weight_evaluation,
        # groups="group" if use_groups else None,
    ).fit(
        train_data=train_data[train_data["location"] == loc].
 ↪drop(columns=["ds"]),
        time_limit=time_limit,
        presets=presets,
        num_stack_levels=num_stack_levels,
        num_bag_folds=num_bag_folds if not use_groups else 2,# just put␣
 ↪somethin, will be overwritten anyways
        num_bag_sets=num_bag_sets,
        tuning_data=tuning_data[tuning_data["location"] == loc].
 ↪reset_index(drop=True).drop(columns=["ds"]) if use_tune_data else None,
        use_bag_holdout=use_bag_holdout,
        # holdout_frac=holdout_frac,
        excluded_model_types=excluded_model_types,
    )

    # evaluate on test data
    if use_test_data:
        # drop sample_weight column
        t = test_data[test_data["location"] == loc]#.
 ↪drop(columns=["sample_weight"])
        perf = predictor.evaluate(t)
        print("Evaluation on test data:")
        print(perf[predictor.eval_metric.name])

    return predictor

loc = "A"
predictors[0] = fit_predictor_for_location(loc)
```

```
Warning: path already exists! This predictor may overwrite an existing
predictor! path="AutogluonModels/submission_110_jorge_A"
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=1, num_bag_folds=8,
num_bag_sets=1
Beginning AutoGluon training …
AutoGluon will save models to "AutogluonModels/submission_110_jorge_A/"
AutoGluon Version:  0.8.1
```

```
Python Version:     3.10.12
Operating System:   Darwin
Platform Machine:   arm64
Platform Version:   Darwin Kernel Version 22.1.0: Sun Oct  9 20:15:09 PDT 2022;
root:xnu-8792.41.9~2/RELEASE_ARM64_T6000
Disk Space Avail:   136.94 GB / 494.38 GB (27.7%)
Train Data Rows:    30934
Train Data Columns: 44
Tuning Data Rows:    1485
Tuning Data Columns: 44
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (5733.42, 0.0, 673.41535, 1195.24)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    2910.15 MB
        Train Data (Original)  Memory Usage: 13.03 MB (0.4% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 2 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…

Training model for location A…

        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 3): ['elevation:m', 'snow_drift:idx',
'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 40 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
```

```
'clear_sky_rad:W', …]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 39 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', ['bool']) :  2 | ['snow_density:kgm3', 'is_estimated']
        0.1s = Fit runtime
        41 features in original data used to generate 41 features in processed
data.
        Train Data (Processed) Memory Usage: 10.18 MB (0.3% of available memory)
Data preprocessing and feature engineering runtime = 0.14s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
AutoGluon will fit 2 stack levels (L1 to L2) …
Excluded models: ['CAT', 'XGB', 'RF'] (Specified by `excluded_model_types`)
Fitting 8 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 …
        -191.231         = Validation score   (-mean_absolute_error)
        0.02s    = Training   runtime
        124.44s  = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 …
```

```
        -192.9182        = Validation score   (-mean_absolute_error)
        0.02s    = Training   runtime
        133.8s   = Validation runtime
Fitting model: LightGBMXT_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
```

```
[1000]  valid_set's l1: 188.903
[2000]  valid_set's l1: 183.38
[3000]  valid_set's l1: 180.745
[4000]  valid_set's l1: 178.855
[5000]  valid_set's l1: 177.036
[6000]  valid_set's l1: 176.178
[7000]  valid_set's l1: 175.372
[8000]  valid_set's l1: 174.791
[9000]  valid_set's l1: 174.317
[10000] valid_set's l1: 173.968
[1000]  valid_set's l1: 195.036
[2000]  valid_set's l1: 190.396
[3000]  valid_set's l1: 187.398
[4000]  valid_set's l1: 185.562
[5000]  valid_set's l1: 184.402
[6000]  valid_set's l1: 183.498
[7000]  valid_set's l1: 182.975
[8000]  valid_set's l1: 182.6
[9000]  valid_set's l1: 182.308
[10000] valid_set's l1: 182.089
[1000]  valid_set's l1: 174.756
[2000]  valid_set's l1: 170.021
[3000]  valid_set's l1: 167.451
[4000]  valid_set's l1: 166.491
[5000]  valid_set's l1: 165.668
[6000]  valid_set's l1: 164.958
[7000]  valid_set's l1: 164.495
[8000]  valid_set's l1: 164.319
[9000]  valid_set's l1: 164.064
[10000] valid_set's l1: 163.91
[1000]  valid_set's l1: 185.905
[2000]  valid_set's l1: 180.669
[3000]  valid_set's l1: 178.641
[4000]  valid_set's l1: 177.158
[5000]  valid_set's l1: 176.004
[6000]  valid_set's l1: 175.196
[7000]  valid_set's l1: 174.998
[8000]  valid_set's l1: 174.683
[9000]  valid_set's l1: 174.314
[10000] valid_set's l1: 174.04
[1000]  valid_set's l1: 183.164
```

```
[2000]  valid_set's l1: 176.218
[3000]  valid_set's l1: 172.938
[4000]  valid_set's l1: 171.415
[5000]  valid_set's l1: 169.986
[6000]  valid_set's l1: 169.039
[7000]  valid_set's l1: 168.349
[8000]  valid_set's l1: 167.719
[9000]  valid_set's l1: 167.298
[10000] valid_set's l1: 166.983
[1000]  valid_set's l1: 170.514
[2000]  valid_set's l1: 165.088
[3000]  valid_set's l1: 162.919
[4000]  valid_set's l1: 161.49
[5000]  valid_set's l1: 160.782
[6000]  valid_set's l1: 160.182
[7000]  valid_set's l1: 159.752
[8000]  valid_set's l1: 159.568
[9000]  valid_set's l1: 159.34
[10000] valid_set's l1: 159.161
[1000]  valid_set's l1: 189.562
[2000]  valid_set's l1: 183.203
[3000]  valid_set's l1: 180.322
[4000]  valid_set's l1: 178.848
[5000]  valid_set's l1: 178.163
[6000]  valid_set's l1: 177.178
[7000]  valid_set's l1: 176.468
[8000]  valid_set's l1: 175.848
[9000]  valid_set's l1: 175.54
[10000] valid_set's l1: 175.128
[1000]  valid_set's l1: 182.64
[2000]  valid_set's l1: 176.705
[3000]  valid_set's l1: 173.867
[4000]  valid_set's l1: 171.977
[5000]  valid_set's l1: 170.847
[6000]  valid_set's l1: 170.152
[7000]  valid_set's l1: 169.313
[8000]  valid_set's l1: 168.821
[9000]  valid_set's l1: 168.33
[10000] valid_set's l1: 167.86
        -85.9423         = Validation score   (-mean_absolute_error)
        632.95s  = Training   runtime
        5.25s    = Validation runtime
Fitting model: LightGBM_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
[1000]  valid_set's l1: 190.81
[2000]  valid_set's l1: 187.978
```

```
[3000]  valid_set's l1: 187.108
[4000]  valid_set's l1: 186.753
[5000]  valid_set's l1: 186.643
[6000]  valid_set's l1: 186.467
[7000]  valid_set's l1: 186.373
[8000]  valid_set's l1: 186.278
[9000]  valid_set's l1: 186.204
[10000] valid_set's l1: 186.123
[1000]  valid_set's l1: 191.986
[2000]  valid_set's l1: 189.445
[3000]  valid_set's l1: 188.988
[4000]  valid_set's l1: 188.636
[5000]  valid_set's l1: 188.683
[1000]  valid_set's l1: 176.636
[2000]  valid_set's l1: 174.634
[3000]  valid_set's l1: 174.348
[4000]  valid_set's l1: 174.119
[5000]  valid_set's l1: 174.113
[1000]  valid_set's l1: 185.029
[2000]  valid_set's l1: 183.879
[3000]  valid_set's l1: 183.05
[4000]  valid_set's l1: 182.497
[5000]  valid_set's l1: 182.009
[6000]  valid_set's l1: 181.735
[7000]  valid_set's l1: 181.671
[8000]  valid_set's l1: 181.593
[9000]  valid_set's l1: 181.573
[10000] valid_set's l1: 181.544
[1000]  valid_set's l1: 181.506
[2000]  valid_set's l1: 179.187
[3000]  valid_set's l1: 178.017
[4000]  valid_set's l1: 177.891
[5000]  valid_set's l1: 177.634
[6000]  valid_set's l1: 177.572
[7000]  valid_set's l1: 177.521
[8000]  valid_set's l1: 177.447
[9000]  valid_set's l1: 177.318
[10000] valid_set's l1: 177.293
[1000]  valid_set's l1: 172.629
[2000]  valid_set's l1: 170.206
[3000]  valid_set's l1: 170.272
[1000]  valid_set's l1: 189.686
[2000]  valid_set's l1: 187.737
[3000]  valid_set's l1: 187.338
[4000]  valid_set's l1: 187.186
[5000]  valid_set's l1: 187.004
[6000]  valid_set's l1: 186.824
[7000]  valid_set's l1: 186.707
```

```
[8000]	valid_set's l1: 186.705
[9000]	valid_set's l1: 186.646
[10000]	valid_set's l1: 186.621
[1000]	valid_set's l1: 185.982
[2000]	valid_set's l1: 183.834
[3000]	valid_set's l1: 183.272
[4000]	valid_set's l1: 182.99
[5000]	valid_set's l1: 182.899
[6000]	valid_set's l1: 182.904
[7000]	valid_set's l1: 182.841
[8000]	valid_set's l1: 182.842
[9000]	valid_set's l1: 182.805
[10000]	valid_set's l1: 182.799

        -90.5139         = Validation score   (-mean_absolute_error)
        548.74s  = Training   runtime
        4.83s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 …
        -102.5659        = Validation score   (-mean_absolute_error)
        5.91s    = Training   runtime
        0.78s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
        -103.2339        = Validation score   (-mean_absolute_error)
        146.92s  = Training   runtime
        0.23s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
        -86.8158         = Validation score   (-mean_absolute_error)
        288.74s  = Training   runtime
        0.16s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy

[1000]	valid_set's l1: 180.725
[2000]	valid_set's l1: 179.262
[3000]	valid_set's l1: 178.983
[4000]	valid_set's l1: 178.905
[5000]	valid_set's l1: 178.876
[6000]	valid_set's l1: 178.871
[7000]	valid_set's l1: 178.868
[8000]	valid_set's l1: 178.867
[9000]	valid_set's l1: 178.866
[10000]	valid_set's l1: 178.866
[1000]	valid_set's l1: 185.8
[2000]	valid_set's l1: 184.475
```

```
[3000]  valid_set's l1: 184.237
[4000]  valid_set's l1: 184.136
[5000]  valid_set's l1: 184.096
[6000]  valid_set's l1: 184.089
[7000]  valid_set's l1: 184.085
[8000]  valid_set's l1: 184.083
[9000]  valid_set's l1: 184.083
[10000] valid_set's l1: 184.083
[1000]  valid_set's l1: 170.845
[2000]  valid_set's l1: 169.438
[3000]  valid_set's l1: 169.218
[4000]  valid_set's l1: 169.166
[5000]  valid_set's l1: 169.138
[6000]  valid_set's l1: 169.138
[7000]  valid_set's l1: 169.136
[8000]  valid_set's l1: 169.135
[9000]  valid_set's l1: 169.134
[10000] valid_set's l1: 169.134
[1000]  valid_set's l1: 180.147
[2000]  valid_set's l1: 179.141
[3000]  valid_set's l1: 178.869
[4000]  valid_set's l1: 178.801
[5000]  valid_set's l1: 178.772
[6000]  valid_set's l1: 178.766
[7000]  valid_set's l1: 178.763
[8000]  valid_set's l1: 178.763
[9000]  valid_set's l1: 178.762
[10000] valid_set's l1: 178.762
[1000]  valid_set's l1: 174.711
[2000]  valid_set's l1: 173.3
[3000]  valid_set's l1: 172.959
[4000]  valid_set's l1: 172.852
[5000]  valid_set's l1: 172.82
[6000]  valid_set's l1: 172.8
[7000]  valid_set's l1: 172.796
[8000]  valid_set's l1: 172.794
[9000]  valid_set's l1: 172.793
[10000] valid_set's l1: 172.792
[1000]  valid_set's l1: 164.749
[2000]  valid_set's l1: 163.662
[3000]  valid_set's l1: 163.349
[4000]  valid_set's l1: 163.261
[5000]  valid_set's l1: 163.24
[6000]  valid_set's l1: 163.233
[7000]  valid_set's l1: 163.23
[8000]  valid_set's l1: 163.229
[9000]  valid_set's l1: 163.229
[10000] valid_set's l1: 163.229
```

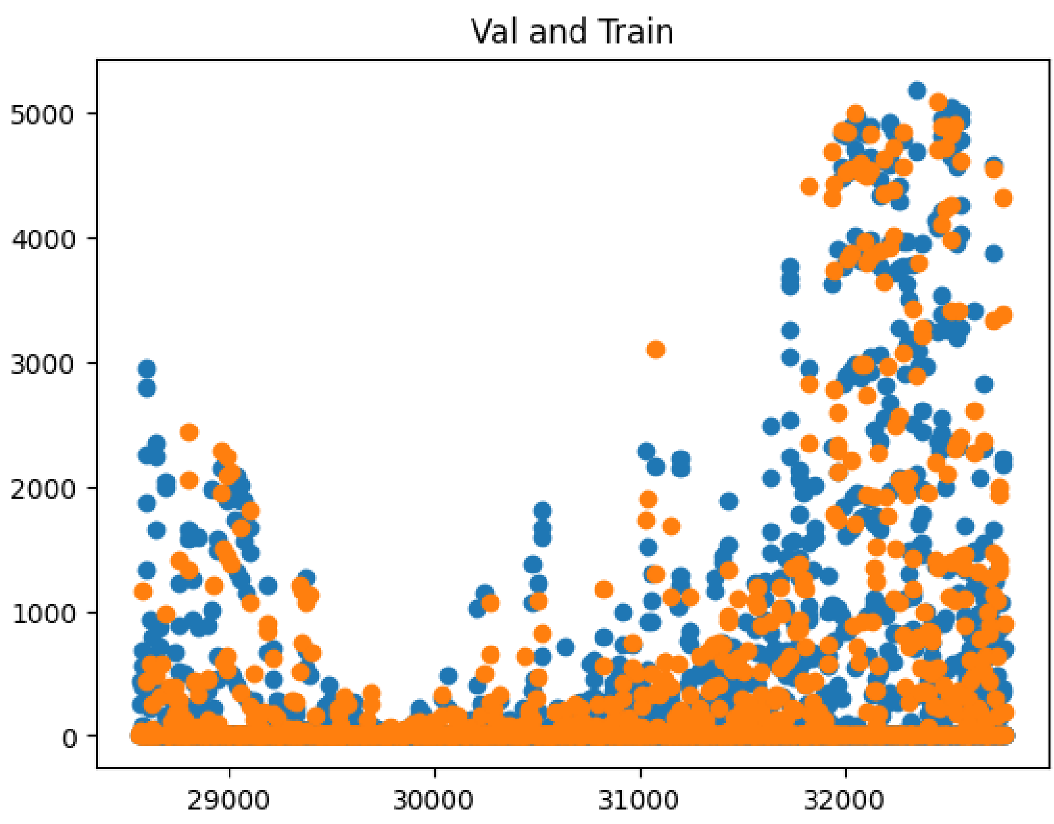
```
[1000]  valid_set's l1: 186.346
[2000]  valid_set's l1: 185.128
[3000]  valid_set's l1: 184.833
[4000]  valid_set's l1: 184.771
[5000]  valid_set's l1: 184.758
[6000]  valid_set's l1: 184.75
[7000]  valid_set's l1: 184.746
[8000]  valid_set's l1: 184.746
[9000]  valid_set's l1: 184.745
[10000] valid_set's l1: 184.745
[1000]  valid_set's l1: 179.754
[2000]  valid_set's l1: 178.455
[3000]  valid_set's l1: 178.101
[4000]  valid_set's l1: 177.99
[5000]  valid_set's l1: 177.949
[6000]  valid_set's l1: 177.937
[7000]  valid_set's l1: 177.934
[8000]  valid_set's l1: 177.931
[9000]  valid_set's l1: 177.93
[10000] valid_set's l1: 177.929
        -87.4067         = Validation score   (-mean_absolute_error)
        2201.24s         = Training   runtime
        18.68s   = Validation runtime
Fitting model: WeightedEnsemble_L2 …
        -82.427  = Validation score   (-mean_absolute_error)
        0.09s    = Training   runtime
        0.0s     = Validation runtime
Excluded models: ['CAT', 'XGB', 'RF'] (Specified by `excluded_model_types`)
Fitting 6 L2 models …
Fitting model: LightGBMXT_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
        -82.1222         = Validation score   (-mean_absolute_error)
        46.41s   = Training   runtime
        0.09s    = Validation runtime
Fitting model: LightGBM_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
        -83.3841         = Validation score   (-mean_absolute_error)
        28.77s   = Training   runtime
        0.05s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L2 …
        -84.3902         = Validation score   (-mean_absolute_error)
        4.63s    = Training   runtime
        0.69s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
```

```
SequentialLocalFoldFittingStrategy
        -84.1655         = Validation score   (-mean_absolute_error)
        129.16s  = Training   runtime
        0.22s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
        -80.4986         = Validation score   (-mean_absolute_error)
        120.72s  = Training   runtime
        0.18s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
        -81.9218         = Validation score   (-mean_absolute_error)
        95.33s   = Training   runtime
        0.11s    = Validation runtime
Fitting model: WeightedEnsemble_L3 …
        -79.952  = Validation score   (-mean_absolute_error)
        0.06s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 4587.76s … Best model:
"WeightedEnsemble_L3"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_110_jorge_A/")
Evaluation: mean_absolute_error on test data: -104.0170346191831
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -104.0170346191831,
    "root_mean_squared_error": -343.2434308293062,
    "mean_squared_error": -117816.0528074727,
    "r2": 0.8150976418318635,
    "pearsonr": 0.908343922594865,
    "median_absolute_error": -0.4310464859008789
}

Evaluation on test data:
-104.0170346191831
```

[113]:

```python
import matplotlib.pyplot as plt
leaderboards = [None, None, None]
def leaderboard_for_location(i, loc):
    if use_tune_data:
        plt.scatter(train_data[(train_data["location"] == loc) &␣
 ↪(train_data["is_estimated"]==True)]["y"].index,␣
 ↪train_data[(train_data["location"] == loc) &␣
 ↪(train_data["is_estimated"]==True)]["y"])
```

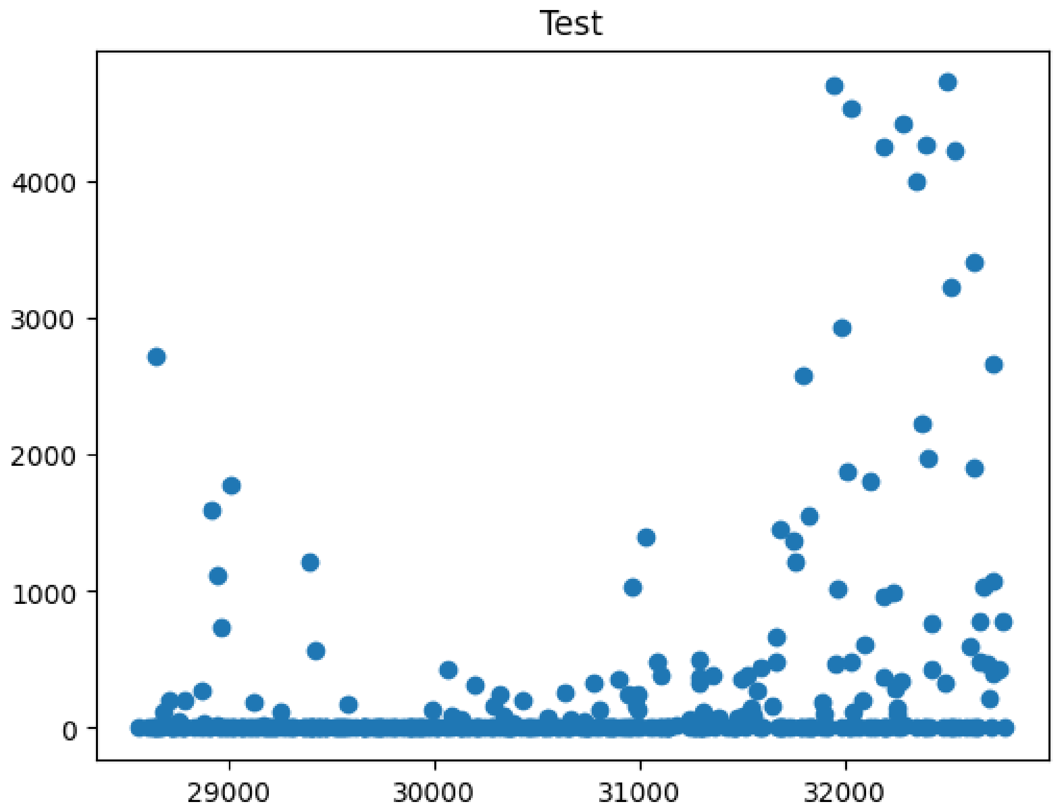
```
        plt.scatter(tuning_data[tuning_data["location"] == loc]["y"].index,␣
↪tuning_data[tuning_data["location"] == loc]["y"])
        plt.title("Val and Train")
        plt.show()

        if use_test_data:
            lb = predictors[i].leaderboard(test_data[test_data["location"] ==␣
↪loc])
            lb["location"] = loc
            plt.scatter(test_data[test_data["location"] == loc]["y"].index,␣
↪test_data[test_data["location"] == loc]["y"])
            plt.title("Test")

            return lb

    return pd.DataFrame()

leaderboards[0] = leaderboard_for_location(0, loc)
```




```
                              model  score_test   score_val  pred_time_test
pred_time_val     fit_time  pred_time_test_marginal  pred_time_val_marginal
```

```
fit_time_marginal  stack_level  can_infer  fit_order
0     LightGBMLarge_BAG_L2 -100.344676  -81.921807        8.954369
288.291218  3919.872742                0.072763              0.108297
95.326025            2       True         15
1          LightGBM_BAG_L2 -103.208461  -83.384102        8.904328
288.237538  3853.321632                0.022722              0.054617
28.774914            2       True         11
2      WeightedEnsemble_L3 -104.017035  -79.951958        9.136753
288.687065  4169.809804                0.001524              0.000200
0.058752            3       True         16
3   NeuralNetFastAI_BAG_L2 -106.006906  -84.165495        8.981432
288.398759  3953.707798                0.099826              0.215838
129.161081            2       True         13
4    NeuralNetTorch_BAG_L2 -106.316233  -80.498632        8.962640
288.362729  3945.263947                0.081034              0.179808
120.717229            2       True         14
5        LightGBMXT_BAG_L1 -106.334331  -85.942263        1.012713
5.252166   632.951649              1.012713              5.252166
632.951649            1       True          3
6     ExtraTreesMSE_BAG_L2 -106.669390  -84.390184        9.144472
288.868813  3829.178929                0.262866              0.685892
4.632211            2       True         12
7      WeightedEnsemble_L2 -107.032810  -82.427021        4.203135
24.093291  3123.019629               0.003368              0.000217
0.087302            2       True          9
8        LightGBMXT_BAG_L2 -107.268312  -82.122217        8.918326
288.277918  3870.960499                0.036720              0.094997
46.413781            2       True         10
9    NeuralNetTorch_BAG_L1 -113.815401  -86.815769        0.065626
0.158559   288.738178              0.065626              0.158559
288.738178            1       True          7
10    LightGBMLarge_BAG_L1 -117.227225  -87.406706        3.121427
18.682350  2201.242500               3.121427             18.682350
2201.242500            1       True          8
11         LightGBM_BAG_L1 -118.139797  -90.513914        0.794818
4.832484   548.740454              0.794818              4.832484
548.740454            1       True          4
12  NeuralNetFastAI_BAG_L1 -121.584755 -103.233884        0.091631
0.234678   146.924149              0.091631              0.234678
146.924149            1       True          6
13    ExtraTreesMSE_BAG_L1 -130.426819 -102.565879        0.311221
0.777855     5.908648              0.311221              0.777855
5.908648            1       True          5
14   KNeighborsDist_BAG_L1 -189.567130 -192.918160        1.621655
133.801945     0.020957                1.621655            133.801945
0.020957            1       True          2
15   KNeighborsUnif_BAG_L1 -191.283846 -191.231007        1.862515
124.442885     0.020183                1.862515            124.442885
```

```
0.020183            1        True           1
```



[114]:
```python
loc = "B"
predictors[1] = fit_predictor_for_location(loc)
leaderboards[1] = leaderboard_for_location(1, loc)
```

```
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=1, num_bag_folds=8,
num_bag_sets=1
Beginning AutoGluon training …
AutoGluon will save models to "AutogluonModels/submission_110_jorge_B/"
AutoGluon Version:  0.8.1
Python Version:     3.10.12
Operating System:   Darwin
Platform Machine:   arm64
Platform Version:   Darwin Kernel Version 22.1.0: Sun Oct  9 20:15:09 PDT 2022;
root:xnu-8792.41.9~2/RELEASE_ARM64_T6000
Disk Space Avail:   133.05 GB / 494.38 GB (26.9%)
Train Data Rows:    27377
Train Data Columns: 44
Tuning Data Rows:    1485
Tuning Data Columns: 44
```

```
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (1152.3, -0.0, 98.11625, 206.48535)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    2927.59 MB
        Train Data (Original)  Memory Usage: 11.6 MB (0.4% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 2 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
```

```
Training model for location B…
```

```
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 41 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 40 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', ['bool']) :  2 | ['snow_density:kgm3', 'is_estimated']
        0.1s = Fit runtime
        42 features in original data used to generate 42 features in processed
data.
        Train Data (Processed) Memory Usage: 9.29 MB (0.3% of available memory)
```

```
Data preprocessing and feature engineering runtime = 0.13s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
AutoGluon will fit 2 stack levels (L1 to L2) …
Excluded models: ['CAT', 'XGB', 'RF'] (Specified by `excluded_model_types`)
Fitting 8 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 …
        -28.5444         = Validation score   (-mean_absolute_error)
        0.02s    = Training   runtime
        93.97s   = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 …
        -28.798  = Validation score   (-mean_absolute_error)
        0.02s    = Training   runtime
        93.0s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy

[1000]  valid_set's l1: 24.6066
[2000]  valid_set's l1: 23.6791
[3000]  valid_set's l1: 23.2765
[4000]  valid_set's l1: 23.0041
```

```
[5000]  valid_set's l1: 22.8351
[6000]  valid_set's l1: 22.703
[7000]  valid_set's l1: 22.6119
[8000]  valid_set's l1: 22.5473
[9000]  valid_set's l1: 22.4997
[10000] valid_set's l1: 22.4547
[1000]  valid_set's l1: 26.1536
[2000]  valid_set's l1: 25.1944
[3000]  valid_set's l1: 24.7599
[4000]  valid_set's l1: 24.5647
[5000]  valid_set's l1: 24.3903
[6000]  valid_set's l1: 24.2417
[7000]  valid_set's l1: 24.1488
[8000]  valid_set's l1: 24.094
[9000]  valid_set's l1: 24.036
[10000] valid_set's l1: 24.0033
[1000]  valid_set's l1: 25.4777
[2000]  valid_set's l1: 24.4918
[3000]  valid_set's l1: 24.0555
[4000]  valid_set's l1: 23.8539
[5000]  valid_set's l1: 23.6537
[6000]  valid_set's l1: 23.5255
[7000]  valid_set's l1: 23.4309
[8000]  valid_set's l1: 23.3571
[9000]  valid_set's l1: 23.2886
[10000] valid_set's l1: 23.2219
[1000]  valid_set's l1: 24.5311
[2000]  valid_set's l1: 23.6487
[3000]  valid_set's l1: 23.2186
[4000]  valid_set's l1: 23.0203
[5000]  valid_set's l1: 22.8983
[6000]  valid_set's l1: 22.7893
[7000]  valid_set's l1: 22.7117
[8000]  valid_set's l1: 22.6492
[9000]  valid_set's l1: 22.5971
[10000] valid_set's l1: 22.5638
[1000]  valid_set's l1: 24.1285
[2000]  valid_set's l1: 23.194
[3000]  valid_set's l1: 22.7303
[4000]  valid_set's l1: 22.5007
[5000]  valid_set's l1: 22.3169
[6000]  valid_set's l1: 22.2316
[7000]  valid_set's l1: 22.1399
[8000]  valid_set's l1: 22.0551
[9000]  valid_set's l1: 22.0141
[10000] valid_set's l1: 21.9924
[1000]  valid_set's l1: 26.5324
[2000]  valid_set's l1: 25.5886
```

```
[3000]  valid_set's l1: 25.1107
[4000]  valid_set's l1: 24.9062
[5000]  valid_set's l1: 24.7015
[6000]  valid_set's l1: 24.5774
[7000]  valid_set's l1: 24.48
[8000]  valid_set's l1: 24.3971
[9000]  valid_set's l1: 24.3528
[10000] valid_set's l1: 24.3183
[1000]  valid_set's l1: 25.087
[2000]  valid_set's l1: 24.2731
[3000]  valid_set's l1: 23.9132
[4000]  valid_set's l1: 23.6529
[5000]  valid_set's l1: 23.497
[6000]  valid_set's l1: 23.3759
[7000]  valid_set's l1: 23.3042
[8000]  valid_set's l1: 23.26
[9000]  valid_set's l1: 23.2203
[10000] valid_set's l1: 23.1934
[1000]  valid_set's l1: 26.1041
[2000]  valid_set's l1: 25.0475
[3000]  valid_set's l1: 24.5287
[4000]  valid_set's l1: 24.1975
[5000]  valid_set's l1: 23.9587
[6000]  valid_set's l1: 23.75
[7000]  valid_set's l1: 23.6282
[8000]  valid_set's l1: 23.5357
[9000]  valid_set's l1: 23.4652
[10000] valid_set's l1: 23.3927
        -13.5737         = Validation score   (-mean_absolute_error)
        522.87s  = Training   runtime
        4.88s    = Validation runtime
Fitting model: LightGBM_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
[1000]  valid_set's l1: 24.9522
[2000]  valid_set's l1: 24.5442
[3000]  valid_set's l1: 24.4594
[4000]  valid_set's l1: 24.4109
[5000]  valid_set's l1: 24.3685
[6000]  valid_set's l1: 24.3412
[7000]  valid_set's l1: 24.3262
[8000]  valid_set's l1: 24.3211
[9000]  valid_set's l1: 24.3099
[10000] valid_set's l1: 24.3097
[1000]  valid_set's l1: 26.4973
[2000]  valid_set's l1: 26.0908
[3000]  valid_set's l1: 25.9096
```

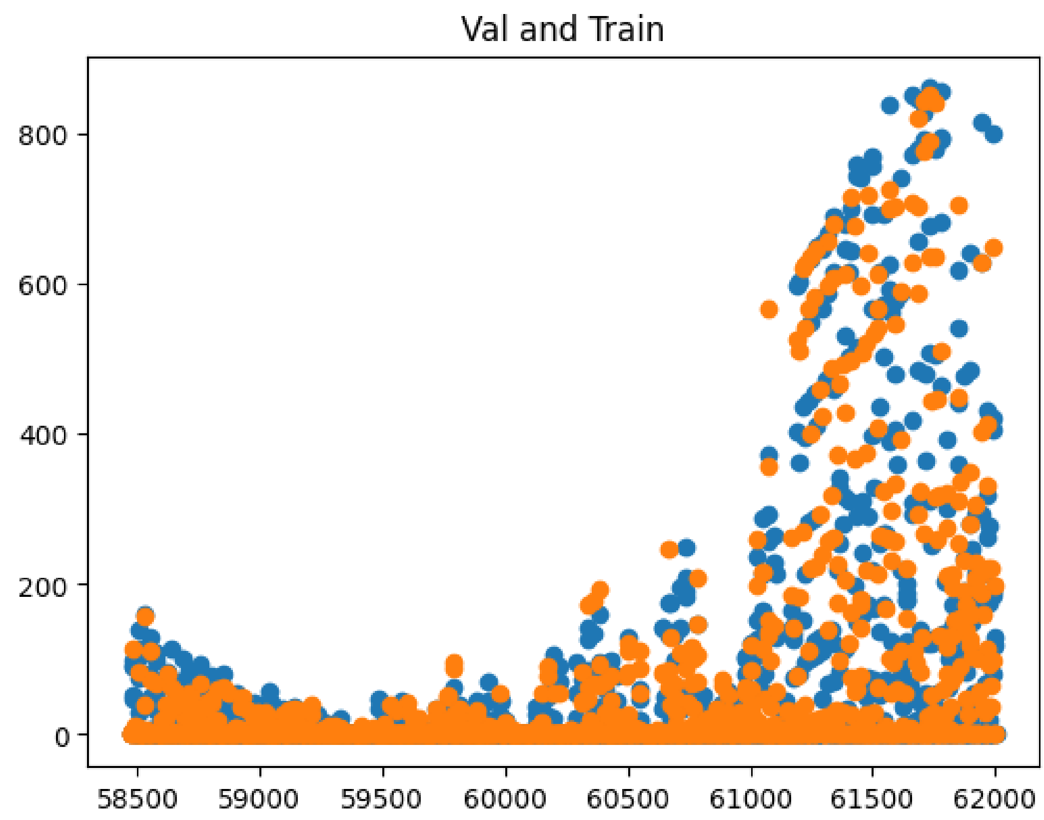
```
[4000]  valid_set's l1: 25.8257
[5000]  valid_set's l1: 25.818
[6000]  valid_set's l1: 25.8016
[7000]  valid_set's l1: 25.7876
[8000]  valid_set's l1: 25.7819
[9000]  valid_set's l1: 25.7801
[10000] valid_set's l1: 25.7784
[1000]  valid_set's l1: 25.8879
[2000]  valid_set's l1: 25.3336
[3000]  valid_set's l1: 25.1701
[4000]  valid_set's l1: 25.0749
[5000]  valid_set's l1: 25.029
[6000]  valid_set's l1: 25.0093
[7000]  valid_set's l1: 24.9925
[8000]  valid_set's l1: 24.983
[9000]  valid_set's l1: 24.9758
[10000] valid_set's l1: 24.9721
[1000]  valid_set's l1: 24.5142
[2000]  valid_set's l1: 24.2196
[3000]  valid_set's l1: 24.1623
[4000]  valid_set's l1: 24.1404
[5000]  valid_set's l1: 24.1173
[6000]  valid_set's l1: 24.1159
[7000]  valid_set's l1: 24.1155
[8000]  valid_set's l1: 24.1091
[9000]  valid_set's l1: 24.1041
[10000] valid_set's l1: 24.104
[1000]  valid_set's l1: 24.2652
[2000]  valid_set's l1: 23.9394
[3000]  valid_set's l1: 23.8498
[4000]  valid_set's l1: 23.8122
[5000]  valid_set's l1: 23.7879
[6000]  valid_set's l1: 23.7667
[7000]  valid_set's l1: 23.7575
[8000]  valid_set's l1: 23.7453
[9000]  valid_set's l1: 23.733
[10000] valid_set's l1: 23.7268
[1000]  valid_set's l1: 27.0443
[2000]  valid_set's l1: 26.6344
[3000]  valid_set's l1: 26.5488
[4000]  valid_set's l1: 26.4874
[5000]  valid_set's l1: 26.4572
[6000]  valid_set's l1: 26.4365
[7000]  valid_set's l1: 26.42
[8000]  valid_set's l1: 26.411
[9000]  valid_set's l1: 26.4075
[10000] valid_set's l1: 26.4086
[1000]  valid_set's l1: 25.5867
```

```
[2000]  valid_set's l1: 25.1833
[3000]  valid_set's l1: 25.0424
[4000]  valid_set's l1: 24.9884
[5000]  valid_set's l1: 24.9564
[6000]  valid_set's l1: 24.9411
[7000]  valid_set's l1: 24.9343
[8000]  valid_set's l1: 24.9313
[9000]  valid_set's l1: 24.9314
[10000] valid_set's l1: 24.93
[1000]  valid_set's l1: 25.6007
[2000]  valid_set's l1: 25.2662
[3000]  valid_set's l1: 25.1166
[4000]  valid_set's l1: 25.0633
[5000]  valid_set's l1: 25.0337
[6000]  valid_set's l1: 25.0034
[7000]  valid_set's l1: 24.9894
[8000]  valid_set's l1: 24.9834
[9000]  valid_set's l1: 24.9783
[10000] valid_set's l1: 24.9755

        -14.6686         = Validation score   (-mean_absolute_error)
        686.91s  = Training   runtime
        5.4s     = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 …
        -15.3912         = Validation score   (-mean_absolute_error)
        3.02s    = Training   runtime
        0.51s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
        -13.3993         = Validation score   (-mean_absolute_error)
        111.86s  = Training   runtime
        0.19s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
        -13.0471         = Validation score   (-mean_absolute_error)
        291.06s  = Training   runtime
        0.14s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy

[1000]  valid_set's l1: 23.657
[2000]  valid_set's l1: 23.5214
[3000]  valid_set's l1: 23.4867
[4000]  valid_set's l1: 23.4801
[5000]  valid_set's l1: 23.4772
[6000]  valid_set's l1: 23.4762
```

```
[7000]  valid_set's l1: 23.476
[8000]  valid_set's l1: 23.4758
[9000]  valid_set's l1: 23.4757
[10000] valid_set's l1: 23.4757
[1000]  valid_set's l1: 25.0907
[2000]  valid_set's l1: 24.9303
[3000]  valid_set's l1: 24.9013
[4000]  valid_set's l1: 24.8949
[5000]  valid_set's l1: 24.8924
[6000]  valid_set's l1: 24.8916
[7000]  valid_set's l1: 24.8912
[8000]  valid_set's l1: 24.891
[9000]  valid_set's l1: 24.891
[10000] valid_set's l1: 24.891
[1000]  valid_set's l1: 24.1339
[2000]  valid_set's l1: 23.9525
[3000]  valid_set's l1: 23.9249
[4000]  valid_set's l1: 23.913
[5000]  valid_set's l1: 23.9098
[6000]  valid_set's l1: 23.9087
[7000]  valid_set's l1: 23.9084
[8000]  valid_set's l1: 23.9083
[9000]  valid_set's l1: 23.9082
[10000] valid_set's l1: 23.9082
[1000]  valid_set's l1: 23.4733
[2000]  valid_set's l1: 23.3321
[3000]  valid_set's l1: 23.3063
[4000]  valid_set's l1: 23.2949
[5000]  valid_set's l1: 23.2926
[6000]  valid_set's l1: 23.2915
[7000]  valid_set's l1: 23.2912
[8000]  valid_set's l1: 23.2911
[9000]  valid_set's l1: 23.2911
[10000] valid_set's l1: 23.291
[1000]  valid_set's l1: 23.4346
[2000]  valid_set's l1: 23.2845
[3000]  valid_set's l1: 23.2629
[4000]  valid_set's l1: 23.2533
[5000]  valid_set's l1: 23.2486
[6000]  valid_set's l1: 23.2472
[7000]  valid_set's l1: 23.2467
[8000]  valid_set's l1: 23.2466
[9000]  valid_set's l1: 23.2465
[10000] valid_set's l1: 23.2465
[1000]  valid_set's l1: 25.7984
[2000]  valid_set's l1: 25.6523
[3000]  valid_set's l1: 25.6247
[4000]  valid_set's l1: 25.6147
```

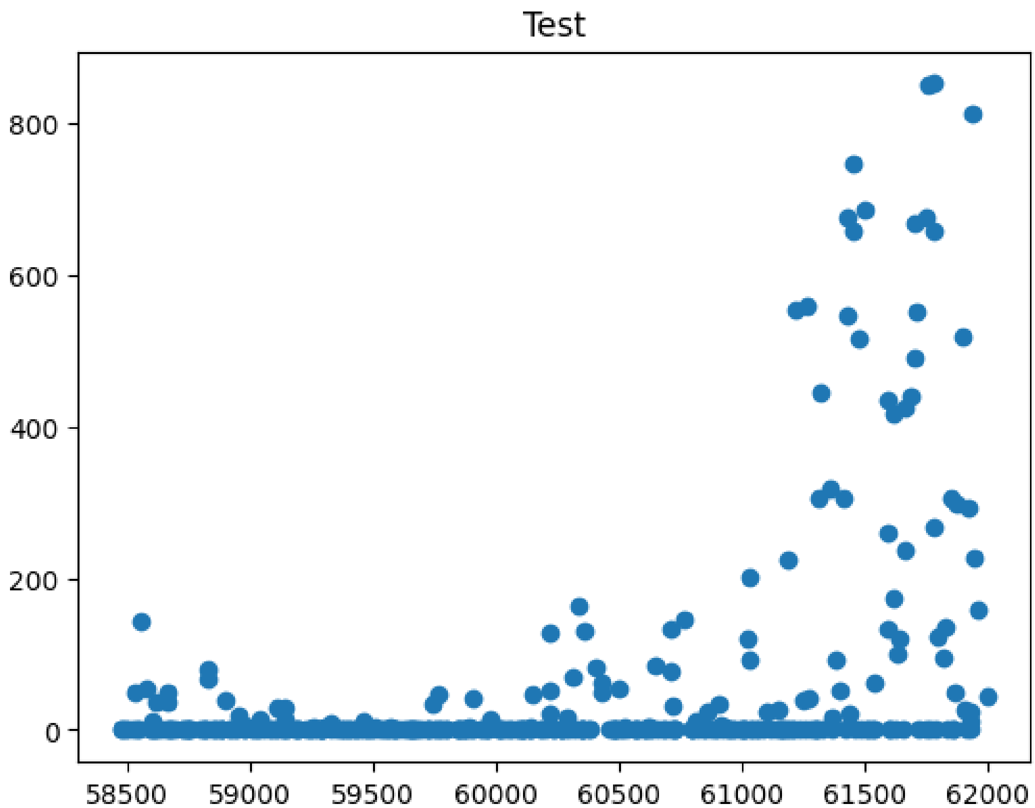
```
[5000]  valid_set's l1: 25.6124
[6000]  valid_set's l1: 25.6116
[7000]  valid_set's l1: 25.6114
[8000]  valid_set's l1: 25.6114
[9000]  valid_set's l1: 25.6113
[10000] valid_set's l1: 25.6113
[1000]  valid_set's l1: 24.1284
[2000]  valid_set's l1: 23.9857
[3000]  valid_set's l1: 23.9582
[4000]  valid_set's l1: 23.9506
[5000]  valid_set's l1: 23.9497
[6000]  valid_set's l1: 23.9493
[7000]  valid_set's l1: 23.9492
[8000]  valid_set's l1: 23.9492
[9000]  valid_set's l1: 23.9493
[1000]  valid_set's l1: 24.8423
[2000]  valid_set's l1: 24.6732
[3000]  valid_set's l1: 24.6384
[4000]  valid_set's l1: 24.6314
[5000]  valid_set's l1: 24.6295
[6000]  valid_set's l1: 24.629
[7000]  valid_set's l1: 24.6288
[8000]  valid_set's l1: 24.6287
[9000]  valid_set's l1: 24.6287
[10000] valid_set's l1: 24.6287
        -14.1202         = Validation score   (-mean_absolute_error)
        2253.9s  = Training   runtime
        15.58s   = Validation runtime
Fitting model: WeightedEnsemble_L2 …
        -12.543  = Validation score   (-mean_absolute_error)
        0.12s    = Training   runtime
        0.0s     = Validation runtime
Excluded models: ['CAT', 'XGB', 'RF'] (Specified by `excluded_model_types`)
Fitting 6 L2 models …
Fitting model: LightGBMXT_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
        -12.7141         = Validation score   (-mean_absolute_error)
        52.76s   = Training   runtime
        0.1s     = Validation runtime
Fitting model: LightGBM_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
        -12.6462         = Validation score   (-mean_absolute_error)
        36.2s    = Training   runtime
        0.06s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L2 …
```

```
        -12.4796        = Validation score   (-mean_absolute_error)
        4.47s    = Training   runtime
        0.6s     = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
        -12.7655        = Validation score   (-mean_absolute_error)
        115.98s  = Training   runtime
        0.19s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
        -12.6433        = Validation score   (-mean_absolute_error)
        168.98s  = Training   runtime
        0.19s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
        -12.4036        = Validation score   (-mean_absolute_error)
        138.53s  = Training   runtime
        0.18s    = Validation runtime
Fitting model: WeightedEnsemble_L3 …
        -12.2989        = Validation score   (-mean_absolute_error)
        0.06s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 4647.27s … Best model:
"WeightedEnsemble_L3"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_110_jorge_B/")
Evaluation: mean_absolute_error on test data: -10.307926520086648
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -10.307926520086648,
    "root_mean_squared_error": -29.011999010962043,
    "mean_squared_error": -841.6960866120626,
    "r2": 0.9637641668722874,
    "pearsonr": 0.9817235510710304,
    "median_absolute_error": -0.06828609108924866
}

Evaluation on test data:
-10.307926520086648
```




```
                     model  score_test  score_val  pred_time_test  pred_time_val
fit_time  pred_time_test_marginal  pred_time_val_marginal  fit_time_marginal
stack_level  can_infer  fit_order
0       WeightedEnsemble_L3  -10.307927 -12.298876        9.243580     214.636314
4181.700541                0.001464                0.000256           0.064059
3        True           16
1      ExtraTreesMSE_BAG_L2  -10.348747 -12.479581        9.051565     214.267819
3874.126700                0.229068                0.603500           4.471227
2        True           12
2      LightGBMLarge_BAG_L2  -10.564845 -12.403603        8.912773     213.844759
4008.183409                0.090276                0.180440         138.527936
2        True           15
3     NeuralNetTorch_BAG_L2  -10.646161 -12.643333        8.922772     213.852118
4038.637319                0.100275                0.187799         168.981846
2        True           14
4         LightGBMXT_BAG_L2  -10.820018 -12.714061        8.867528     213.763518
3922.419844                0.045031                0.099200          52.764372
2        True           10
5       WeightedEnsemble_L2  -10.925381 -12.543013        1.496924       5.720273
928.928778               0.003038                0.000434           0.119027
2        True            9
```

```
6          LightGBM_BAG_L2  -11.038288 -12.646211        8.848975     213.725104
3905.851579                 0.026478                0.060785          36.196106
2       True          11
7          LightGBM_BAG_L1  -11.107067 -14.668551        1.169937       5.397365
686.906615                 1.169937                5.397365         686.906615
1       True           4
8        LightGBMXT_BAG_L1  -11.242241 -13.573695        1.145201       4.878384
522.867838                 1.145201                4.878384         522.867838
1       True           3
9     LightGBMLarge_BAG_L1  -11.277426 -14.120209        3.382947      15.576877
2253.900146                 3.382947               15.576877        2253.900146
1       True           8
10  NeuralNetFastAI_BAG_L2  -11.489746 -12.765498        8.920271     213.854777
3985.634665                 0.097774                0.190458         115.979192
2       True          13
11   NeuralNetTorch_BAG_L1  -11.683837 -13.047096        0.067755       0.138829
291.064770                 0.067755                0.138829         291.064770
1       True           7
12  NeuralNetFastAI_BAG_L1  -12.510399 -13.399319        0.090306       0.187633
111.860070                 0.090306                0.187633         111.860070
1       True           6
13    ExtraTreesMSE_BAG_L1  -13.140435 -15.391153        0.190624       0.514992
3.017073                 0.190624                0.514992           3.017073
1       True           5
14   KNeighborsDist_BAG_L1  -23.570593 -28.797984        1.122122      92.997750
0.019037                 1.122122               92.997750           0.019037
1       True           2
15   KNeighborsUnif_BAG_L1  -24.697224 -28.544405        1.653605      93.972488
0.019923                 1.653605               93.972488           0.019923
1       True           1
```



[115]:
```
loc = "C"
predictors[2] = fit_predictor_for_location(loc)
leaderboards[2] = leaderboard_for_location(2, loc)
```

```
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=1, num_bag_folds=8,
num_bag_sets=1
Beginning AutoGluon training …
AutoGluon will save models to "AutogluonModels/submission_110_jorge_C/"
AutoGluon Version:  0.8.1
Python Version:     3.10.12
Operating System:   Darwin
Platform Machine:   arm64
Platform Version:   Darwin Kernel Version 22.1.0: Sun Oct  9 20:15:09 PDT 2022;
root:xnu-8792.41.9~2/RELEASE_ARM64_T6000
Disk Space Avail:   129.83 GB / 494.38 GB (26.3%)
Train Data Rows:    24073
Train Data Columns: 44
Tuning Data Rows:     1481
Tuning Data Columns: 44
Label Column: y
```

```
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and label-values can't be converted to int).
        Label info (max, min, mean, stddev): (999.6, -0.0, 80.87539, 169.67845)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    2783.4 MB
        Train Data (Original)  Memory Usage: 10.27 MB (0.4% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 2 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…

Training model for location C…

        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 3): ['elevation:m', 'snow_drift:idx',
'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 40 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 39 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', ['bool']) :  2 | ['snow_density:kgm3', 'is_estimated']
        0.1s = Fit runtime
        41 features in original data used to generate 41 features in processed
data.
        Train Data (Processed) Memory Usage: 8.02 MB (0.3% of available memory)
```

```
Data preprocessing and feature engineering runtime = 0.11s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
AutoGluon will fit 2 stack levels (L1 to L2) …
Excluded models: ['CAT', 'XGB', 'RF'] (Specified by `excluded_model_types`)
Fitting 8 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 …
        -19.8149         = Validation score   (-mean_absolute_error)
        0.02s    = Training   runtime
        69.44s   = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 …
        -20.1923         = Validation score   (-mean_absolute_error)
        0.02s    = Training   runtime
        74.58s   = Validation runtime
Fitting model: LightGBMXT_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy

[1000]  valid_set's l1: 20.7202
[2000]  valid_set's l1: 19.7905
[3000]  valid_set's l1: 19.4751
[4000]  valid_set's l1: 19.2767
```

```
[5000]  valid_set's l1: 19.1244
[6000]  valid_set's l1: 19.0403
[7000]  valid_set's l1: 18.9539
[8000]  valid_set's l1: 18.9012
[9000]  valid_set's l1: 18.8691
[10000] valid_set's l1: 18.8531
[1000]  valid_set's l1: 19.2997
[2000]  valid_set's l1: 18.6743
[3000]  valid_set's l1: 18.3347
[4000]  valid_set's l1: 18.1588
[5000]  valid_set's l1: 18.0626
[6000]  valid_set's l1: 18.0199
[7000]  valid_set's l1: 17.9583
[8000]  valid_set's l1: 17.9319
[9000]  valid_set's l1: 17.8987
[10000] valid_set's l1: 17.8951
[1000]  valid_set's l1: 19.2787
[2000]  valid_set's l1: 18.5285
[3000]  valid_set's l1: 18.2341
[4000]  valid_set's l1: 18.0164
[5000]  valid_set's l1: 17.89
[6000]  valid_set's l1: 17.8061
[7000]  valid_set's l1: 17.7362
[8000]  valid_set's l1: 17.6911
[9000]  valid_set's l1: 17.647
[10000] valid_set's l1: 17.6257
[1000]  valid_set's l1: 18.1031
[2000]  valid_set's l1: 17.4878
[3000]  valid_set's l1: 17.1303
[4000]  valid_set's l1: 16.9477
[5000]  valid_set's l1: 16.7992
[6000]  valid_set's l1: 16.698
[7000]  valid_set's l1: 16.647
[8000]  valid_set's l1: 16.6108
[9000]  valid_set's l1: 16.5933
[10000] valid_set's l1: 16.5608
[1000]  valid_set's l1: 20.4047
[2000]  valid_set's l1: 19.6667
[3000]  valid_set's l1: 19.3143
[4000]  valid_set's l1: 19.1268
[5000]  valid_set's l1: 18.995
[6000]  valid_set's l1: 18.9557
[7000]  valid_set's l1: 18.9164
[8000]  valid_set's l1: 18.8828
[9000]  valid_set's l1: 18.877
[10000] valid_set's l1: 18.854
[1000]  valid_set's l1: 18.7347
[2000]  valid_set's l1: 18.1887
```

```
[3000]  valid_set's l1: 17.9766
[4000]  valid_set's l1: 17.8678
[5000]  valid_set's l1: 17.7846
[6000]  valid_set's l1: 17.7443
[7000]  valid_set's l1: 17.7236
[8000]  valid_set's l1: 17.6999
[9000]  valid_set's l1: 17.7009
[10000] valid_set's l1: 17.6843
[1000]  valid_set's l1: 19.9884
[2000]  valid_set's l1: 19.4274
[3000]  valid_set's l1: 19.133
[4000]  valid_set's l1: 18.9602
[5000]  valid_set's l1: 18.8547
[6000]  valid_set's l1: 18.79
[7000]  valid_set's l1: 18.734
[8000]  valid_set's l1: 18.7127
[9000]  valid_set's l1: 18.6797
[10000] valid_set's l1: 18.6481
[1000]  valid_set's l1: 19.671
[2000]  valid_set's l1: 19.0659
[3000]  valid_set's l1: 18.7509
[4000]  valid_set's l1: 18.5232
[5000]  valid_set's l1: 18.406
[6000]  valid_set's l1: 18.2896
[7000]  valid_set's l1: 18.2117
[8000]  valid_set's l1: 18.1624
[9000]  valid_set's l1: 18.1272
[10000] valid_set's l1: 18.0972
        -11.8239         = Validation score   (-mean_absolute_error)
        565.56s  = Training   runtime
        4.4s     = Validation runtime
Fitting model: LightGBM_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
[1000]  valid_set's l1: 21.1695
[2000]  valid_set's l1: 20.8856
[3000]  valid_set's l1: 20.8058
[4000]  valid_set's l1: 20.7642
[5000]  valid_set's l1: 20.7467
[6000]  valid_set's l1: 20.7383
[7000]  valid_set's l1: 20.7316
[8000]  valid_set's l1: 20.7275
[9000]  valid_set's l1: 20.7274
[10000] valid_set's l1: 20.7245
[1000]  valid_set's l1: 19.272
[2000]  valid_set's l1: 19.0298
[3000]  valid_set's l1: 18.966
```

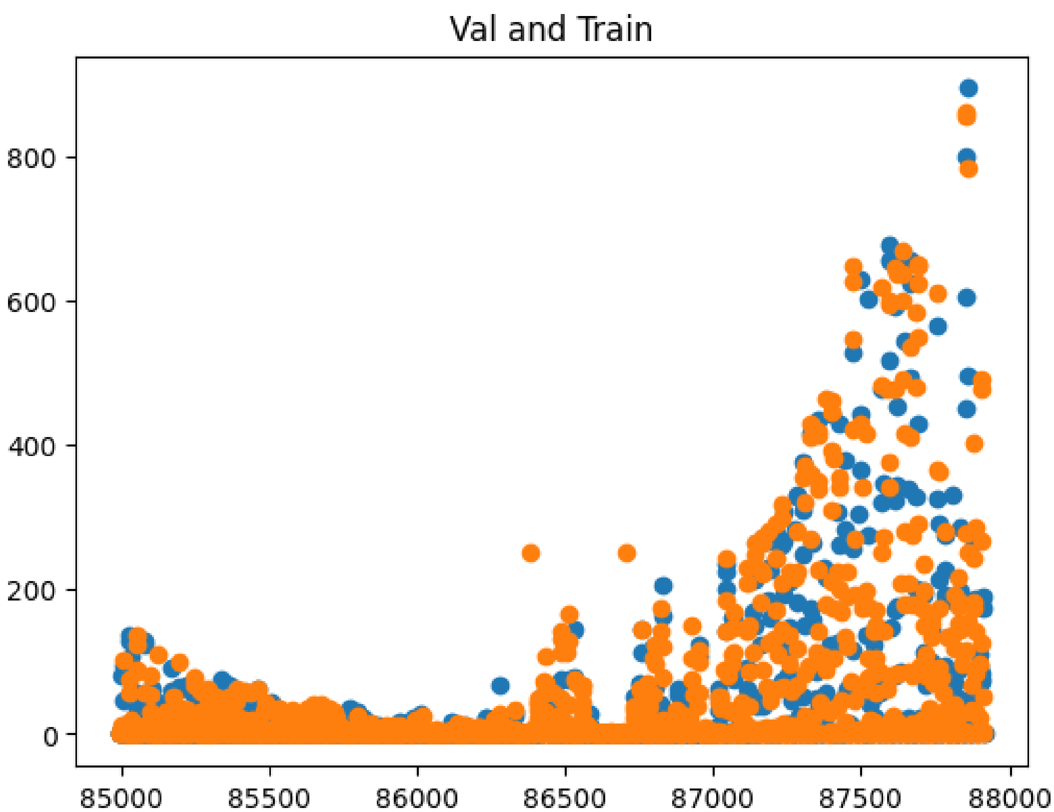
```
[4000]  valid_set's l1: 18.9213
[5000]  valid_set's l1: 18.9005
[6000]  valid_set's l1: 18.8859
[7000]  valid_set's l1: 18.8787
[8000]  valid_set's l1: 18.8736
[9000]  valid_set's l1: 18.8712
[10000] valid_set's l1: 18.8699
[1000]  valid_set's l1: 19.3825
[2000]  valid_set's l1: 19.0735
[3000]  valid_set's l1: 18.984
[4000]  valid_set's l1: 18.95
[5000]  valid_set's l1: 18.9369
[6000]  valid_set's l1: 18.9349
[7000]  valid_set's l1: 18.9331
[8000]  valid_set's l1: 18.9315
[9000]  valid_set's l1: 18.9336
[10000] valid_set's l1: 18.9346
[1000]  valid_set's l1: 18.1294
[2000]  valid_set's l1: 17.8275
[3000]  valid_set's l1: 17.7655
[4000]  valid_set's l1: 17.707
[5000]  valid_set's l1: 17.6861
[6000]  valid_set's l1: 17.6827
[7000]  valid_set's l1: 17.6783
[8000]  valid_set's l1: 17.6746
[9000]  valid_set's l1: 17.6712
[10000] valid_set's l1: 17.6718
[1000]  valid_set's l1: 21.0994
[2000]  valid_set's l1: 20.8712
[3000]  valid_set's l1: 20.7661
[4000]  valid_set's l1: 20.7267
[5000]  valid_set's l1: 20.7043
[6000]  valid_set's l1: 20.695
[7000]  valid_set's l1: 20.6879
[8000]  valid_set's l1: 20.6892
[9000]  valid_set's l1: 20.6906
[1000]  valid_set's l1: 19.6614
[2000]  valid_set's l1: 19.4417
[3000]  valid_set's l1: 19.3371
[4000]  valid_set's l1: 19.3031
[5000]  valid_set's l1: 19.2926
[6000]  valid_set's l1: 19.2892
[7000]  valid_set's l1: 19.2825
[8000]  valid_set's l1: 19.28
[9000]  valid_set's l1: 19.2793
[10000] valid_set's l1: 19.2773
[1000]  valid_set's l1: 20.949
[2000]  valid_set's l1: 20.6894
```

```
[3000]  valid_set's l1: 20.5357
[4000]  valid_set's l1: 20.4757
[5000]  valid_set's l1: 20.4468
[6000]  valid_set's l1: 20.4272
[7000]  valid_set's l1: 20.4193
[8000]  valid_set's l1: 20.4134
[9000]  valid_set's l1: 20.409
[10000] valid_set's l1: 20.408
[1000]  valid_set's l1: 20.0499
[2000]  valid_set's l1: 19.7422
[3000]  valid_set's l1: 19.671
[4000]  valid_set's l1: 19.6066
[5000]  valid_set's l1: 19.5797
[6000]  valid_set's l1: 19.5567
[7000]  valid_set's l1: 19.5496
[8000]  valid_set's l1: 19.544
[9000]  valid_set's l1: 19.5423
[10000] valid_set's l1: 19.5402
        -12.8555         = Validation score   (-mean_absolute_error)
        612.14s  = Training   runtime
        5.24s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 …
        -15.4133         = Validation score   (-mean_absolute_error)
        2.78s    = Training   runtime
        0.49s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
        -13.7096         = Validation score   (-mean_absolute_error)
        106.91s  = Training   runtime
        0.18s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
        -13.452  = Validation score   (-mean_absolute_error)
        180.68s  = Training   runtime
        0.13s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
[1000]  valid_set's l1: 20.16
[2000]  valid_set's l1: 20.0628
[3000]  valid_set's l1: 20.0462
[4000]  valid_set's l1: 20.043
[5000]  valid_set's l1: 20.0421
[6000]  valid_set's l1: 20.0419
[7000]  valid_set's l1: 20.0417
```

```
[8000]  valid_set's l1: 20.0417
[9000]  valid_set's l1: 20.0417
[10000] valid_set's l1: 20.0417
[1000]  valid_set's l1: 18.8487
[2000]  valid_set's l1: 18.7574
[3000]  valid_set's l1: 18.7444
[4000]  valid_set's l1: 18.7424
[5000]  valid_set's l1: 18.7416
[6000]  valid_set's l1: 18.7413
[7000]  valid_set's l1: 18.7412
[8000]  valid_set's l1: 18.7412
[9000]  valid_set's l1: 18.7412
[10000] valid_set's l1: 18.7411
[1000]  valid_set's l1: 19.0558
[2000]  valid_set's l1: 18.9809
[3000]  valid_set's l1: 18.9663
[4000]  valid_set's l1: 18.9639
[5000]  valid_set's l1: 18.9631
[6000]  valid_set's l1: 18.9628
[7000]  valid_set's l1: 18.9628
[8000]  valid_set's l1: 18.9628
[1000]  valid_set's l1: 17.8053
[2000]  valid_set's l1: 17.689
[3000]  valid_set's l1: 17.6689
[4000]  valid_set's l1: 17.6651
[5000]  valid_set's l1: 17.6642
[6000]  valid_set's l1: 17.6639
[7000]  valid_set's l1: 17.6637
[8000]  valid_set's l1: 17.6637
[9000]  valid_set's l1: 17.6637
[10000] valid_set's l1: 17.6637
[1000]  valid_set's l1: 20.5146
[2000]  valid_set's l1: 20.4094
[3000]  valid_set's l1: 20.3954
[4000]  valid_set's l1: 20.3927
[5000]  valid_set's l1: 20.3923
[6000]  valid_set's l1: 20.3921
[7000]  valid_set's l1: 20.3922
[8000]  valid_set's l1: 20.3921
[1000]  valid_set's l1: 18.3672
[2000]  valid_set's l1: 18.2415
[3000]  valid_set's l1: 18.2263
[4000]  valid_set's l1: 18.2226
[5000]  valid_set's l1: 18.2222
[6000]  valid_set's l1: 18.2221
[7000]  valid_set's l1: 18.222
[8000]  valid_set's l1: 18.222
[9000]  valid_set's l1: 18.222
```

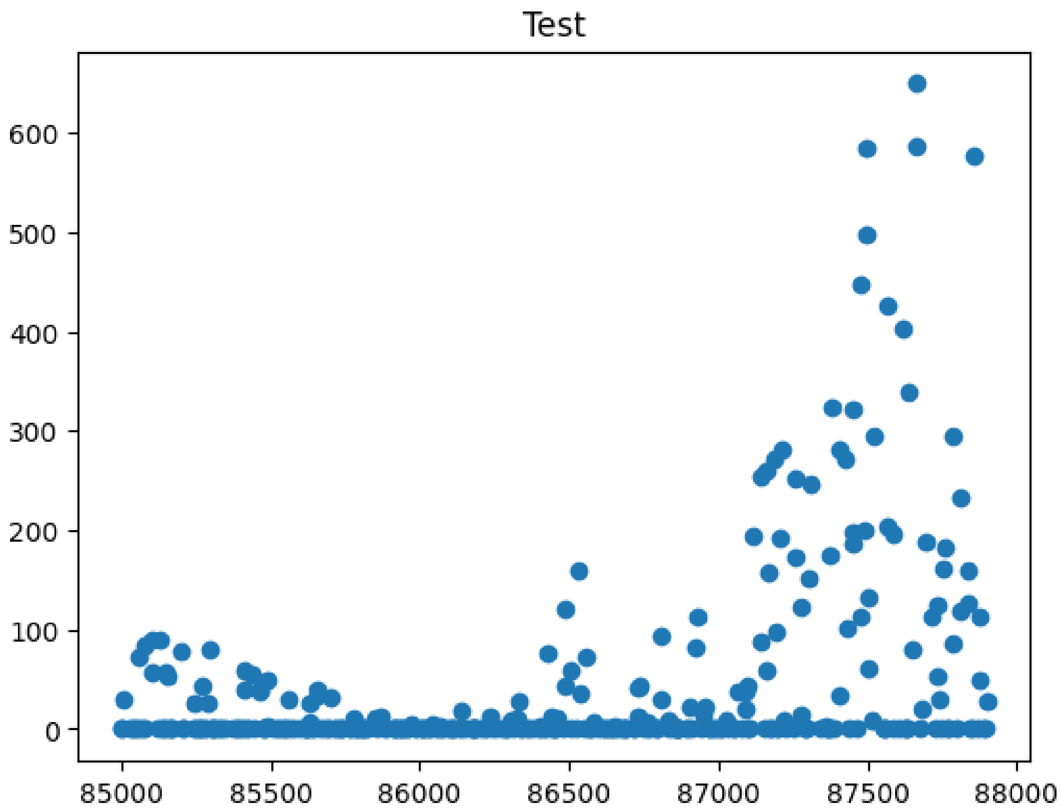
```
[10000] valid_set's l1: 18.222
[1000]  valid_set's l1: 19.9819
[2000]  valid_set's l1: 19.9053
[3000]  valid_set's l1: 19.8844
[4000]  valid_set's l1: 19.8795
[5000]  valid_set's l1: 19.8782
[6000]  valid_set's l1: 19.8778
[7000]  valid_set's l1: 19.8776
[8000]  valid_set's l1: 19.8775
[9000]  valid_set's l1: 19.8775
[10000] valid_set's l1: 19.8775
[1000]  valid_set's l1: 19.3809
[2000]  valid_set's l1: 19.2765
[3000]  valid_set's l1: 19.267
[4000]  valid_set's l1: 19.2637
[5000]  valid_set's l1: 19.263
[6000]  valid_set's l1: 19.2627
[7000]  valid_set's l1: 19.2626
[8000]  valid_set's l1: 19.2626
[9000]  valid_set's l1: 19.2626
[10000] valid_set's l1: 19.2626
        -12.9072         = Validation score   (-mean_absolute_error)
        2053.29s         = Training   runtime
        11.31s   = Validation runtime
Fitting model: WeightedEnsemble_L2 …
        -11.6011         = Validation score   (-mean_absolute_error)
        0.07s    = Training   runtime
        0.0s     = Validation runtime
Excluded models: ['CAT', 'XGB', 'RF'] (Specified by `excluded_model_types`)
Fitting 6 L2 models …
Fitting model: LightGBMXT_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
        -11.9269         = Validation score   (-mean_absolute_error)
        51.27s   = Training   runtime
        0.12s    = Validation runtime
Fitting model: LightGBM_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
        -11.5667         = Validation score   (-mean_absolute_error)
        29.86s   = Training   runtime
        0.06s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L2 …
        -11.6856         = Validation score   (-mean_absolute_error)
        3.78s    = Training   runtime
        0.45s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L2 …
```

```
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
        -11.683  = Validation score   (-mean_absolute_error)
        205.96s  = Training   runtime
        0.28s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
        -11.8348         = Validation score   (-mean_absolute_error)
        255.61s  = Training   runtime
        0.34s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
SequentialLocalFoldFittingStrategy
        -11.6193         = Validation score   (-mean_absolute_error)
        125.75s  = Training   runtime
        0.15s    = Validation runtime
Fitting model: WeightedEnsemble_L3 …
        -11.2382         = Validation score   (-mean_absolute_error)
        0.06s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 4403.32s … Best model:
"WeightedEnsemble_L3"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_110_jorge_C/")
Evaluation: mean_absolute_error on test data: -12.24963412673937
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -12.24963412673937,
    "root_mean_squared_error": -28.97460346825213,
    "mean_squared_error": -839.5276461424484,
    "r2": 0.9149022976011298,
    "pearsonr": 0.9576447734056741,
    "median_absolute_error": -0.4168833941221237
}

Evaluation on test data:
-12.24963412673937
```




```
                         model  score_test  score_val  pred_time_test  pred_time_val
fit_time  pred_time_test_marginal  pred_time_val_marginal  fit_time_marginal
stack_level  can_infer  fit_order
0       WeightedEnsemble_L2  -12.055240 -11.601131        7.948154      74.146465
853.237182                 0.006042                0.000189           0.074040
2        True          9
1    NeuralNetFastAI_BAG_L2  -12.060522 -11.682960       19.442498     166.048577
3727.354681                 0.103819                0.282578         205.958438
2        True         13
2         LightGBMXT_BAG_L1  -12.167139 -11.823881        1.369323       4.401356
565.555315                 1.369323                4.401356         565.555315
1        True          3
3       WeightedEnsemble_L3  -12.249634 -11.238192       19.830340     167.039090
4142.408657                 0.001584                0.000177           0.060789
3        True         16
4      LightGBMLarge_BAG_L2  -12.319525 -11.619336       19.416313     165.914274
3647.148485                 0.077634                0.148275         125.752242
2        True         15
5         LightGBMXT_BAG_L2  -12.365628 -11.926902       19.384327     165.882907
3572.663433                 0.045648                0.116908          51.267189
2        True         10
```

```
6     ExtraTreesMSE_BAG_L2  -12.383183 -11.685594      19.541551     166.213868
3525.171480                0.202872               0.447869          3.775237
2       True          12
7          LightGBM_BAG_L2  -12.439301 -11.566725      19.365296     165.823769
3551.255076                0.026617               0.057770         29.858833
2       True          11
8    NeuralNetTorch_BAG_L2  -12.729589 -11.834786      19.417814     166.102421
3777.003118                0.079135               0.336422        255.606875
2       True          14
9   NeuralNetFastAI_BAG_L1  -13.076601 -13.709620       0.088990       0.175622
106.914660               0.088990               0.175622        106.914660
1       True           6
10   NeuralNetTorch_BAG_L1  -13.081879 -13.452034       0.060264       0.129216
180.677912               0.060264               0.129216        180.677912
1       True           7
11         LightGBM_BAG_L1  -13.515463 -12.855472       1.383632       5.237588
612.141511               1.383632               5.237588        612.141511
1       True           4
12    LightGBMLarge_BAG_L1  -14.282341 -12.907215       3.187670      11.311880
2053.293738                3.187670              11.311880       2053.293738
1       True           8
13    ExtraTreesMSE_BAG_L1  -15.370085 -15.413307       0.149299       0.493397
2.781358                 0.149299               0.493397        2.781358
1       True           5
14   KNeighborsUnif_BAG_L1  -20.049167 -19.814903       6.423535      69.440082
0.015255                 6.423535              69.440082        0.015255
1       True           1
15   KNeighborsDist_BAG_L1  -20.130194 -20.192291       6.675966      74.576858
0.016495                 6.675966              74.576858        0.016495
1       True           2
```



[126]:
```
# save leaderboards to csv
pd.concat(leaderboards).to_csv(f"leaderboards/{new_filename}.csv")

for i in range(len(predictors)):
    print(f"Predictor {i}:")
    print(predictors[i].
 ↪info()["model_info"]["WeightedEnsemble_L3"]["children_info"]["S1F1"]["model_weights"])
print(predictors[0].info())
```

```
Predictor 0:
{'NeuralNetFastAI_BAG_L2': 0.038461538461538464, 'NeuralNetTorch_BAG_L2':
0.7115384615384616, 'LightGBMLarge_BAG_L2': 0.25}
Predictor 1:
{'ExtraTreesMSE_BAG_L2': 0.21052631578947367, 'NeuralNetTorch_BAG_L2':
0.3157894736842105, 'LightGBMLarge_BAG_L2': 0.47368421052631576}
Predictor 2:
```

[ ]:

5 Submit

[117]:
```
import pandas as pd
import matplotlib.pyplot as plt

future_test_data = TabularDataset('X_test_raw.csv')
future_test_data["ds"] = pd.to_datetime(future_test_data["ds"])
#test_data
```

```
Loaded data from: X_test_raw.csv | Columns = 45 / 45 | Rows = 4608 -> 4608
```

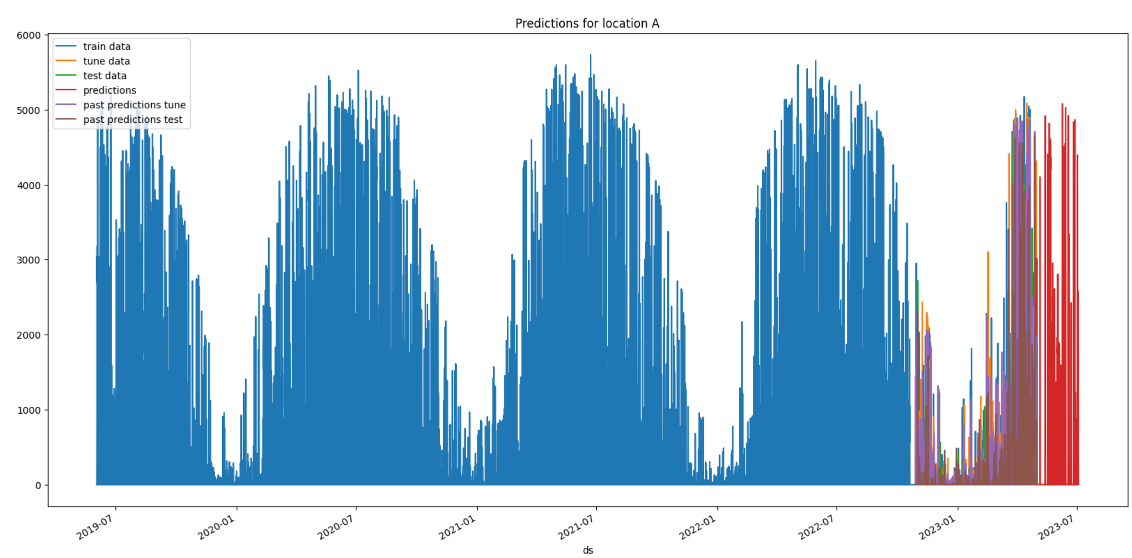
[118]:
```
test_ids = TabularDataset('test.csv')
test_ids["time"] = pd.to_datetime(test_ids["time"])
# merge test_data with test_ids
future_test_data_merged = pd.merge(future_test_data, test_ids, how="inner",␣
↪right_on=["time", "location"], left_on=["ds", "location"])

#test_data_merged
```

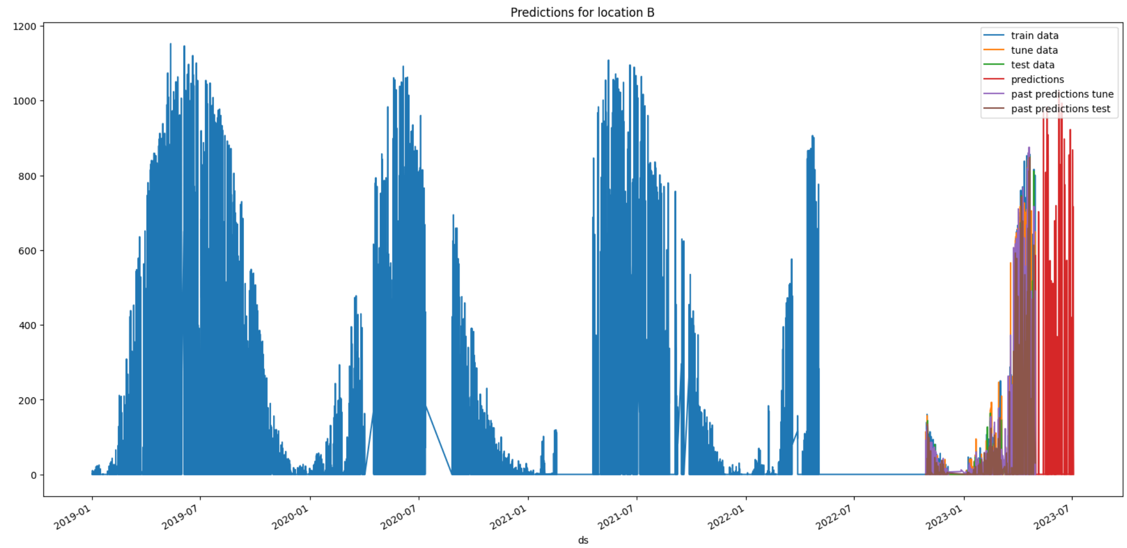
```
Loaded data from: test.csv | Columns = 4 / 4 | Rows = 2160 -> 2160
```

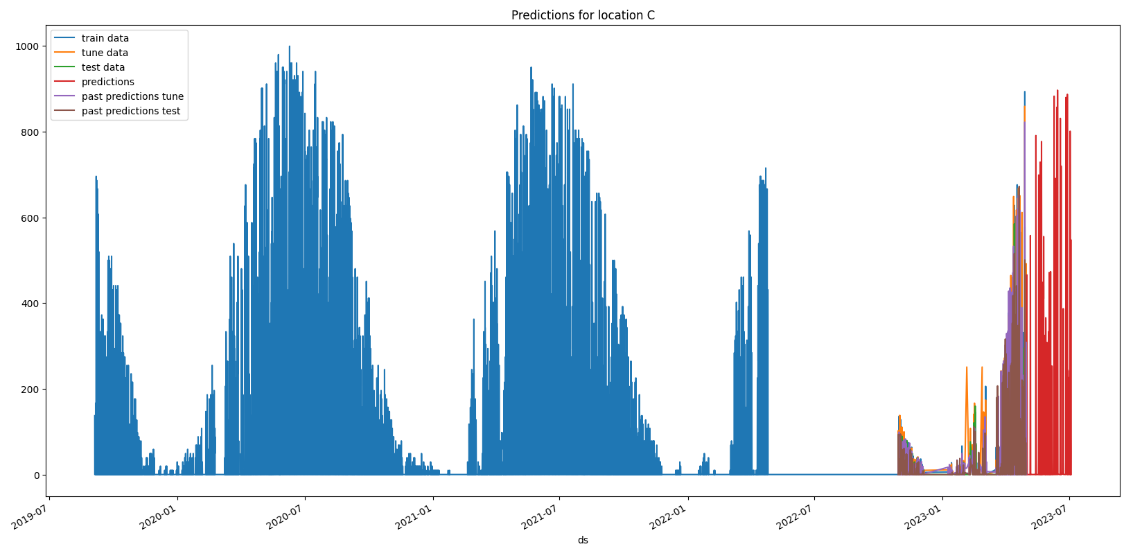
[119]:
```
# predict, grouped by location
predictions = []
location_map = {
    "A": 0,
    "B": 1,
    "C": 2
}
for loc, group in future_test_data.groupby('location'):
    i = location_map[loc]
    subset = future_test_data_merged[future_test_data_merged["location"] ==␣
↪loc].reset_index(drop=True)
    #print(subset)
    pred = predictors[i].predict(subset)
    subset["prediction"] = pred
    predictions.append(subset)

    # get past predictions
    #train_data.loc[train_data["location"] == loc, "prediction"] = ␣
↪predictors[i].predict(train_data[train_data["location"] == loc])
    if use_tune_data:
        tuning_data.loc[tuning_data["location"] == loc, "prediction"] = ␣
↪predictors[i].predict(tuning_data[tuning_data["location"] == loc])
    if use_test_data:
        test_data.loc[test_data["location"] == loc, "prediction"] = ␣
↪predictors[i].predict(test_data[test_data["location"] == loc])
```

[120]:

```
# plot predictions for location A, in addition to train data for A
for loc, idx in location_map.items():
    fig, ax = plt.subplots(figsize=(20, 10))
    # plot train data
    train_data[train_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
 ↪label="train data")
    if use_tune_data:
        tuning_data[tuning_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
 ↪label="tune data")
    if use_test_data:
        test_data[test_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
 ↪label="test data")

    # plot predictions
    predictions[idx].plot(x='ds', y='prediction', ax=ax, label="predictions")

    # plot past predictions
    #train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',␣
 ↪y='prediction', ax=ax, label="past predictions")
    #train_data[train_data["location"]==loc].plot(x='ds', y='prediction',␣
 ↪ax=ax, label="past predictions train")
    if use_tune_data:
        tuning_data[tuning_data["location"]==loc].plot(x='ds', y='prediction',␣
 ↪ax=ax, label="past predictions tune")
    if use_test_data:
        test_data[test_data["location"]==loc].plot(x='ds', y='prediction',␣
 ↪ax=ax, label="past predictions test")


    # title
    ax.set_title(f"Predictions for location {loc}")
```







[121]:
```
temp_predictions = [prediction.copy() for prediction in predictions]
if clip_predictions:
    # clip predictions smaller than 0 to 0
    for pred in temp_predictions:
        # print smallest prediction
        print("Smallest prediction:", pred["prediction"].min())
        pred.loc[pred["prediction"] < 0, "prediction"] = 0
        print("Smallest prediction after clipping:", pred["prediction"].min())
```

```
    # Instead of clipping, shift all prediction values up by the largest negative␣
 ↪number.
    # This way, the smallest prediction will be 0.
    elif shift_predictions:
        for pred in temp_predictions:
            # print smallest prediction
            print("Smallest prediction:", pred["prediction"].min())
            pred["prediction"] = pred["prediction"] - pred["prediction"].min()
            print("Smallest prediction after clipping:", pred["prediction"].min())

    elif shift_predictions_by_average_of_negatives_then_clip:
        for pred in temp_predictions:
            # print smallest prediction
            print("Smallest prediction:", pred["prediction"].min())
            mean_negative = pred[pred["prediction"] < 0]["prediction"].mean()
            # if not nan
            if mean_negative == mean_negative:
                pred["prediction"] = pred["prediction"] - mean_negative

            pred.loc[pred["prediction"] < 0, "prediction"] = 0
            print("Smallest prediction after clipping:", pred["prediction"].min())



# concatenate predictions
submissions_df = pd.concat(temp_predictions)
submissions_df = submissions_df[["id", "prediction"]]
submissions_df
```

```
Smallest prediction: -0.99808645
Smallest prediction after clipping: 0.0
Smallest prediction: -0.2856902
Smallest prediction after clipping: 0.0
Smallest prediction: -0.18184212
Smallest prediction after clipping: 0.0
```

[121]:

	id	prediction
0	0	0.694677
1	1	0.291773
2	2	0.305286
3	3	14.924707
4	4	278.054565
..	…	…
715	2155	68.906456
716	2156	38.547287
717	2157	9.081526

```
718  2158    0.673626
719  2159    0.249410

[2160 rows x 2 columns]
```

[122]:
```
# Save the submission DataFrame to submissions folder, create new name based on␣
↪last submission, format is submission_<last_submission_number + 1>.csv

# Save the submission
print(f"Saving submission to submissions/{new_filename}.csv")
submissions_df.to_csv(os.path.join('submissions', f"{new_filename}.csv"),␣
↪index=False)
print("jall1a")
```

```
Saving submission to submissions/submission_110_jorge.csv
jall1a
```

[123]:
```
# feature importance
# print starting calculating feature importance for location A with big text␣
↪font
print("\033[1m" + "Calculating feature importance for location A..." +␣
↪"\033[0m")
predictors[0].feature_importance(feature_stage="original",␣
↪data=test_data[test_data["location"] == "A"], time_limit=60*10)
print("\033[1m" + "Calculating feature importance for location B..." +␣
↪"\033[0m")
predictors[1].feature_importance(feature_stage="original",␣
↪data=test_data[test_data["location"] == "B"], time_limit=60*10)
print("\033[1m" + "Calculating feature importance for location C..." +␣
↪"\033[0m")
predictors[2].feature_importance(feature_stage="original",␣
↪data=test_data[test_data["location"] == "C"], time_limit=60*10)
```

```
These features in provided data are not utilized by the predictor and will be
ignored: ['ds', 'elevation:m', 'snow_drift:idx', 'location', 'prediction']
Computing feature importance via permutation shuffling for 41 features using 361
rows with 10 shuffle sets… Time limit: 600s…
```

Calculating feature importance for location A…

```
        9326.46s        = Expected runtime (932.65s per shuffle set)
        563.99s = Actual runtime (Completed 1 of 10 shuffle sets) (Early
stopping due to lack of time…)
These features in provided data are not utilized by the predictor and will be
ignored: ['ds', 'elevation:m', 'location', 'prediction']
Computing feature importance via permutation shuffling for 42 features using 361
rows with 10 shuffle sets… Time limit: 600s…
```

Calculating feature importance for location B…

```
        9023.94s        = Expected runtime (902.39s per shuffle set)
        546.63s = Actual runtime (Completed 1 of 10 shuffle sets) (Early
stopping due to lack of time…)
These features in provided data are not utilized by the predictor and will be
ignored: ['ds', 'elevation:m', 'snow_drift:idx', 'location', 'prediction']
Computing feature importance via permutation shuffling for 41 features using 360
rows with 10 shuffle sets… Time limit: 600s…
```

Calculating feature importance for location C…

```
        7542.29s        = Expected runtime (754.23s per shuffle set)
        469.54s = Actual runtime (Completed 1 of 10 shuffle sets) (Early
stopping due to lack of time…)
```

[123]:

	importance	stddev	p_value	n	p99_high	\
clear_sky_rad:W	11.566056	NaN	NaN	1	NaN	
clear_sky_energy_1h:J	7.343914	NaN	NaN	1	NaN	
sun_elevation:d	5.360511	NaN	NaN	1	NaN	
sun_azimuth:d	2.975047	NaN	NaN	1	NaN	
direct_rad:W	2.824565	NaN	NaN	1	NaN	
diffuse_rad:W	2.321137	NaN	NaN	1	NaN	
diffuse_rad_1h:J	2.157632	NaN	NaN	1	NaN	
total_cloud_cover:p	1.618280	NaN	NaN	1	NaN	
direct_rad_1h:J	1.500727	NaN	NaN	1	NaN	
fresh_snow_24h:cm	1.367420	NaN	NaN	1	NaN	
snow_water:kgm2	0.965320	NaN	NaN	1	NaN	
effective_cloud_cover:p	0.924933	NaN	NaN	1	NaN	
relative_humidity_1000hPa:p	0.899903	NaN	NaN	1	NaN	
cloud_base_agl:m	0.630297	NaN	NaN	1	NaN	
visibility:m	0.579226	NaN	NaN	1	NaN	
precip_5min:mm	0.576286	NaN	NaN	1	NaN	
is_day:idx	0.546159	NaN	NaN	1	NaN	
ceiling_height_agl:m	0.395922	NaN	NaN	1	NaN	
air_density_2m:kgm3	0.384625	NaN	NaN	1	NaN	
precip_type_5min:idx	0.305891	NaN	NaN	1	NaN	
pressure_50m:hPa	0.250655	NaN	NaN	1	NaN	
snow_density:kgm3	0.199200	NaN	NaN	1	NaN	
dew_or_rime:idx	0.168742	NaN	NaN	1	NaN	
is_in_shadow:idx	0.137126	NaN	NaN	1	NaN	
super_cooled_liquid_water:kgm2	0.126909	NaN	NaN	1	NaN	
sfc_pressure:hPa	0.117637	NaN	NaN	1	NaN	
prob_rime:p	0.064115	NaN	NaN	1	NaN	
wind_speed_10m:ms	0.061069	NaN	NaN	1	NaN	
rain_water:kgm2	0.042338	NaN	NaN	1	NaN	
absolute_humidity_2m:gm3	0.040718	NaN	NaN	1	NaN	
snow_melt_10min:mm	0.023413	NaN	NaN	1	NaN	
dew_point_2m:K	0.013342	NaN	NaN	1	NaN	
pressure_100m:hPa	0.012425	NaN	NaN	1	NaN	

snow_depth:cm	0.006352	NaN	NaN	1	NaN
is_estimated	0.000000	NaN	NaN	1	NaN
msl_pressure:hPa	-0.042553	NaN	NaN	1	NaN
fresh_snow_1h:cm	-0.237276	NaN	NaN	1	NaN
fresh_snow_6h:cm	-0.453016	NaN	NaN	1	NaN
fresh_snow_3h:cm	-0.457368	NaN	NaN	1	NaN
fresh_snow_12h:cm	-0.482274	NaN	NaN	1	NaN
t_1000hPa:K	-0.491355	NaN	NaN	1	NaN

	p99_low
clear_sky_rad:W	NaN
clear_sky_energy_1h:J	NaN
sun_elevation:d	NaN
sun_azimuth:d	NaN
direct_rad:W	NaN
diffuse_rad:W	NaN
diffuse_rad_1h:J	NaN
total_cloud_cover:p	NaN
direct_rad_1h:J	NaN
fresh_snow_24h:cm	NaN
snow_water:kgm2	NaN
effective_cloud_cover:p	NaN
relative_humidity_1000hPa:p	NaN
cloud_base_agl:m	NaN
visibility:m	NaN
precip_5min:mm	NaN
is_day:idx	NaN
ceiling_height_agl:m	NaN
air_density_2m:kgm3	NaN
precip_type_5min:idx	NaN
pressure_50m:hPa	NaN
snow_density:kgm3	NaN
dew_or_rime:idx	NaN
is_in_shadow:idx	NaN
super_cooled_liquid_water:kgm2	NaN
sfc_pressure:hPa	NaN
prob_rime:p	NaN
wind_speed_10m:ms	NaN
rain_water:kgm2	NaN
absolute_humidity_2m:gm3	NaN
snow_melt_10min:mm	NaN
dew_point_2m:K	NaN
pressure_100m:hPa	NaN
snow_depth:cm	NaN
is_estimated	NaN
msl_pressure:hPa	NaN
fresh_snow_1h:cm	NaN

```
fresh_snow_6h:cm                    NaN
fresh_snow_3h:cm                    NaN
fresh_snow_12h:cm                   NaN
t_1000hPa:K                         NaN
```

[124]:
```
# save this notebook to submissions folder
import subprocess
import os
#subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
 ↪join('notebook_pdfs', f"{new_filename}_automatic_save.pdf"),␣
 ↪"autogluon_each_location.ipynb"])
subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
 ↪join('notebook_pdfs', f"{new_filename}.pdf"), "autogluon_each_location.
 ↪ipynb"])
```

```
[NbConvertApp] Converting notebook autogluon_each_location.ipynb to pdf
[NbConvertApp] Support files will be in
notebook_pdfs/submission_110_jorge_files/
[NbConvertApp] Making directory
./notebook_pdfs/submission_110_jorge_files/notebook_pdfs
[NbConvertApp] Writing 281529 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 1987204 bytes to notebook_pdfs/submission_110_jorge.pdf
```

[124]:
```
CompletedProcess(args=['jupyter', 'nbconvert', '--to', 'pdf', '--output',
'notebook_pdfs/submission_110_jorge.pdf', 'autogluon_each_location.ipynb'],
returncode=0)
```

[125]:
```
# import subprocess

# def execute_git_command(directory, command):
#     """Execute a Git command in the specified directory."""
#     try:
#         result = subprocess.check_output(['git', '-C', directory] + command,␣
 ↪stderr=subprocess.STDOUT)
#         return result.decode('utf-8').strip(), True
#     except subprocess.CalledProcessError as e:
#         print(f"Git command failed with message: {e.output.decode('utf-8').
 ↪strip()}")
#         return e.output.decode('utf-8').strip(), False

# git_repo_path = "."
```

```
# execute_git_command(git_repo_path, ['config', 'user.email',␣
↪'henrikskog01@gmail.com'])
# execute_git_command(git_repo_path, ['config', 'user.name', hello if hello is␣
↪not None else 'Henrik eller Jørgen'])

# branch_name = new_filename

# # add datetime to branch name
# branch_name += f"_{pd.Timestamp.now().strftime('%Y-%m-%d_%H-%M-%S')}"

# commit_msg = "run result"

# execute_git_command(git_repo_path, ['checkout', '-b',branch_name])

# # Navigate to your repo and commit changes
# execute_git_command(git_repo_path, ['add', '.'])
# execute_git_command(git_repo_path, ['commit', '-m',commit_msg])

# # Push to remote
# output, success = execute_git_command(git_repo_path, ['push',␣
↪'origin',branch_name])

# # If the push fails, try setting an upstream branch and push again
# if not success and 'upstream' in output:
#     print("Attempting to set upstream and push again...")
#     execute_git_command(git_repo_path, ['push', '--set-upstream',␣
↪'origin',branch_name])
#     execute_git_command(git_repo_path, ['push', 'origin', 'henrik_branch'])

# execute_git_command(git_repo_path, ['checkout', 'main'])
```

[ ]:

```

```